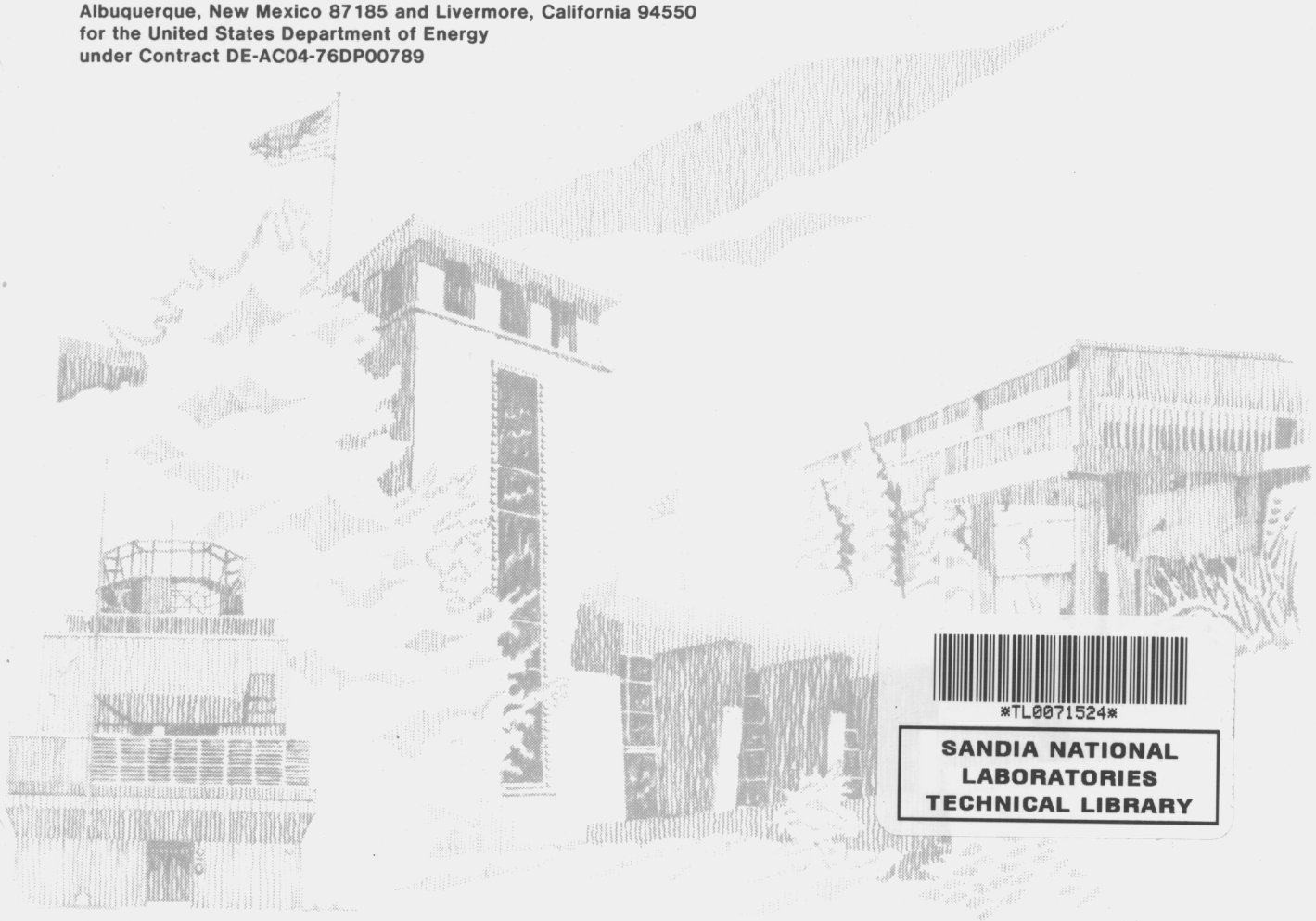
SANDIA REPORT

SAND85—2347 • UC—703
Unlimited Release
Printed June 1992

Sandia Software Guidelines

Volume 4: Configuration Management

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-76DP00789



Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A06
Microfiche copy: A01

Distribution
Category UC-703

SAND85-2347
Unlimited Release
Printed June 1992

Sandia Software Guidelines

Volume 4

Configuration Management

Sandia National Laboratories
Albuquerque, New Mexico 87185

Abstract

This volume is one in a series of *Sandia Software Guidelines* for use in producing quality software within Sandia National Laboratories. This volume is based on the IEEE standard and guide for software configuration management. The basic concepts and detailed guidance on implementation of these concepts are discussed for several software project types. Example planning documents for both projects and organizations are included.

Foreword

This volume is one in a series of *Sandia Software Guidelines* for use in producing quality software within Sandia National Laboratories (SNL). These guidelines, when used in conjunction with IEEE standards and current software engineering methodologies, will help ensure that software developed within SNL is usable, reliable, understandable, maintainable, and portable. When complete, the series will consist of the following documents:

- Volume 1, ***Software Quality Planning*** (SAND85-2344)

  Presents an overview of procedures designed to ensure software quality. Includes a sample software quality assurance plan for a generic Sandia software project.

- Volume 2, ***Documentation*** (SAND85-2345)

  Presents a description of documents needed for developing, maintaining, and defining software projects. Includes sample document outlines.

- Volume 3, ***Standards, Practices, and Conventions*** (SAND85-2346)

  Presents consensus standards and practices for developing and maintaining quality software at SNL. Includes recommended deliverables for major phases of the software life cycle.

- Volume 4, ***Configuration Management*** (SAND85-2347)

  Presents a discussion of configuration management objectives and approaches throughout the software life cycle for software projects at SNL.

- Volume 5, ***Tools, Techniques, and Methodologies*** (SAND85-2348)

  Presents descriptions and a directory of software tools and methodologies available to SNL personnel.

Acknowledgment

A consensus document, like this volume of the *Sandia Software Guidelines*, cannot be produced without the cooperation and hard work of a great many people throughout the Laboratories. The sponsoring Software Quality and Reliability Department wishes to thank the members of the working group who contributed to Volume 4, as well as the members of the balloting group who reviewed and refined it.

Working Group Members, Volume 4

Brandon Ahrens	(2725)	Randy Summers	(6418)
David Bradley (former Sandian)	(6429)	Dave Peercy	(0326)
Olin Bray	(2818)	Eric Tomlin, Co-Editor	(0326)
Blase Gaude	(5012)	Sharon Trauth, Co-Editor	(2545)

Contents

List of Figures

1 Introduction

As much as we may all prefer stability, change is ever present in the world in which we live. Software is no exception. In fact, software might very well be called "change-ware," since its nature is characterized by change. On countless projects we hear statements like, "Hey, that's great, but can you make it do..." and "I know what the problem is; I can fix it in no time." Such seems to be the way of life when it comes to software.

But changes in software can easily get out of hand. Here are just a few examples.

- Last minute changes in the requirements are made. Software changes are implemented, but there is not enough time to update all the documentation.

- Testing uncovers a minor problem, and it takes a revision of only one statement to make it right. The test is rerun to show that the problem is fixed, and the software is delivered. Later, the customer reports a problem. Investigation reveals that the one-line change affected another function elsewhere in the software.

- Delivery time is fast approaching, but there is one feature that just does not work as it should. The customer agrees to take the software without this feature, on a promise that it will be included in a subsequent product revision. The feature is deleted from the software for shipment. Subsequently, the customer reports a problem. The software designer discovers during troubleshooting that not all the code for the deleted feature was actually removed.

- A new programmer on a large project decides that a particular data value is not necessary for the remainder of a module's calculations. He deletes this part of the calculation, finishes his module, and verifies that it works properly. Other programmers, however, need this data value for their modules, and wind up spending three days trying to find out why their modules do not calculate correct answers.

Certainly many more examples come to mind. Time constraints create an environment in which mistakes easily happen. But, as in the second and fourth examples, mistakes happen even under less hectic circumstances. For software to have the elusive characteristic of "quality," we cannot afford to take change for granted. Change will occur, but we must do whatever we can to be sure that the quality of the software product is not jeopardized.

1.1 Intent

Configuration Management (CM) is a familiar concept to many of those involved in hardware engineering environments. CM establishes a comprehensive set of activities that are intended to capture the essence of the design and its definition at any point in time. Configuration Management also provides a basic framework that facilitates change approval, implementation, and tracking throughout the life of the product.

This, the fourth volume in the set of *Sandia Software Guidelines*, presents concepts in Software Configuration Management (SCM), as derived from the familiar hardware world, and illustrate how they are applied to software projects in general. The volume discusses these concepts in the context of their importance to software and software quality, as well as to integrated hardware-software systems. Volume 4 centers on SCM techniques that can be applied to the kinds of software projects found at Sandia National Laboratories.

This volume is not intended to provide the answers to all questions in the area of configuration management and its application to software. Rather, it is intended to stimulate all individuals concerned with software — developers, managers, and quality personnel — to think about the problems and issues involved in the effective management of software change.

Based on the consensus standards developed and published by the Institute of Electrical and Electronics Engineers (IEEE), this volume provides information that should supplement the content of the related IEEE standards [IEE90-b]*, [IEE87], and assist the user in the application of these nationally recognized standards to their particular projects.

1.2 Environment

A quick scan of the organizations listed in the Sandia Directory reveals the diversity of functions performed throughout the Laboratories. The environment found throughout Sandia is seen as one of innovation and change. Whether research or production-oriented design and development, Sandia projects are forging new frontiers in the state-of-the-art. Change and iteration are day-to-day parts of doing business.

The principles and techniques necessary to transform software development from a "creative art" to a "structured science" are also in a state of rapid evolution. A cursory review of the professional community nationwide indicates an increase in the attention and concern being given to the software engineering field. Countless seminars and classes are available, as well as new texts, methodologies, techniques, and tools.

The software engineering environment at Sandia is changing with that of the national arena. Engineering Procedures (EPs) are periodically revised; new areas of concern are being identified. This document addresses the fundamental principles and concepts that should remain applicable throughout any upcoming advances.

1.3 Applicability

The material presented in this volume is intended to be applicable to the variety of software projects at Sandia, although specific mention of every kind of application is not attempted.

Most projects at Sandia can be regarded as either "weapon" or "non-weapon" activities, the former encompassing all projects undertaken in support of the design and development of nuclear

* References for additional information are marked throughout this volume by brackets in this way: [XYZ89] ([XYZ89-x] for different references of the same author and year) and [SSGvn] for *Sandia Software Guidelines*.

weapon systems. Thus, discussions of SCM techniques are divided, when necessary, into these two basic categories — war reserve (WR) and non-war reserve (non-WR) — applications, respectively.

Where differences in approaches do exist, the material is first presented in general, philosophical, and conceptual terms. This information is followed by examples for both War Reserve and non-War Reserve projects to illustrate how SCM might be applied in each environment.

This document is a set of guidelines, not directives. Enforcement of an application of these guidelines should be established at the project level (or alternatively, at the department or directorate level), with complete management support. Each project leader should consciously decide how to adopt these guidelines and implement them for the particular project. Additional references are given to provide the technical detail necessary to assist in this determination. Once an understanding of the specific activities necessary for the project is reached, Software Configuration Management procedures should be documented and rigorously followed.

This guide is designed to be used with the other volumes of the *Sandia Software Guidelines,* references [SSGv1], [SSGv3], and [SSGv5]. These volumes provide details on preferred software quality engineering practices at Sandia National Laboratories. The reader is also encouraged to consult the IEEE software engineering standards and guides for further information. The IEEE standards are available through the Sandia Design Information Center, the Sandia Technical Library, or from the IEEE Computer Society. The IEEE also provides a single, bound edition of current software engineering standards [IEE91].

1.4 Guideline Organization

Volume 4 is divided into five chapters and six appendices to help clarify SCM principles. This first chapter provides general information.

Chapter 2 is a short tutorial on SCM. It presents the objectives and benefits of SCM and establishes the foundation for the remainder of Volume 4, providing information about the specific functions involved in applying configuration management principles to software. This chapter discusses selection and identification of software components that will be controlled. Also covered is the controlling of changes made to the selected software components. It outlines the importance of tracking the status of these components, the changes made to them, and the interrelationships among them throughout the life cycle. Chapter 2 also covers the review and audit process for both (1) the final software product — to be that sure everything is internally consistent, up to date, and ready for release, and (2) the SCM process itself — to be sure that it is both implemented and effective. Finally, this chapter discusses how SCM integrates into the software life cycle.

Chapter 3 covers the necessary, but often neglected, subject of planning. It presents two approaches to planning SCM: planning at the organizational and project levels. Techniques for simplifying the planning process, such as standards, guides, and procedures, are discussed, along with some pointers that can help the reader prepare a more effective approach to SCM.

Chapter 4 provides additional detail by presenting several plan concepts and some specific techniques that can help implement of an effective SCM program. It discusses the use of libraries

for the controlled retention of software components. It also instructs the reader about how Software Configuration Control Boards (SCCBs) can be utilized to approve and govern the implementation of changes, and to assure all concerned are properly notified of a pending change. This chapter discusses how approval levels for proposed changes can vary according to the stage of project completion or the severity of the impact of the change. Finally, the chapter alerts the reader to some issues that should be addressed both during development and after release of the software to assure that an effective SCM system is established.

Chapter 5 discusses the use of tools to support the SCM process. It begins by addressing the goals of automating a software configuration management system, followed by a framework for evaluating tools. A detailed structure of what an automated system should be able to control is presented. The chapter concludes with a discussion of concerns that cannot be resolved today with automation, and which can be addressed only by thorough planning.

Finally, six appendices are included for reference and to provide examples of Software Configuration Management Plans (SCMPs). Appendix A contains references, and Appendix B contains the glossary and acronyms.

Appendix C is a thematic for an organizational SCMP. Appendix D is a sample plan for an embedded weapons application. Appendix E is a sample configuration management plan and procedures for a large research application. Finally, Appendix F is a template for a configuration management plan. The template is designed to be adaptable to weapon as well as non-weapon application, and reimbursable projects. This template is available from the Software Quality and Reliability Engineering Department on disk (Word for Windows and ASCII) and on-line via the Sandia Vax System.

1.5 How to Use This Volume

Figure 1-1 is a guide to the reader for locating information concerning (1) the main activities of software configuration management and (2) the level of planning recommended for a software development project or an organization's software development activities. The reader should keep in mind that Section 2 is for the developer or manager who is <u>not</u> familiar with software configuration management. Section 3 is for the reader who is <u>not</u> familiar with SCM planning. Section 4 recommends specific levels of configuration management and control for software development and support. Section 5 is for the reader interested in what an SCM tool should be able to manage.

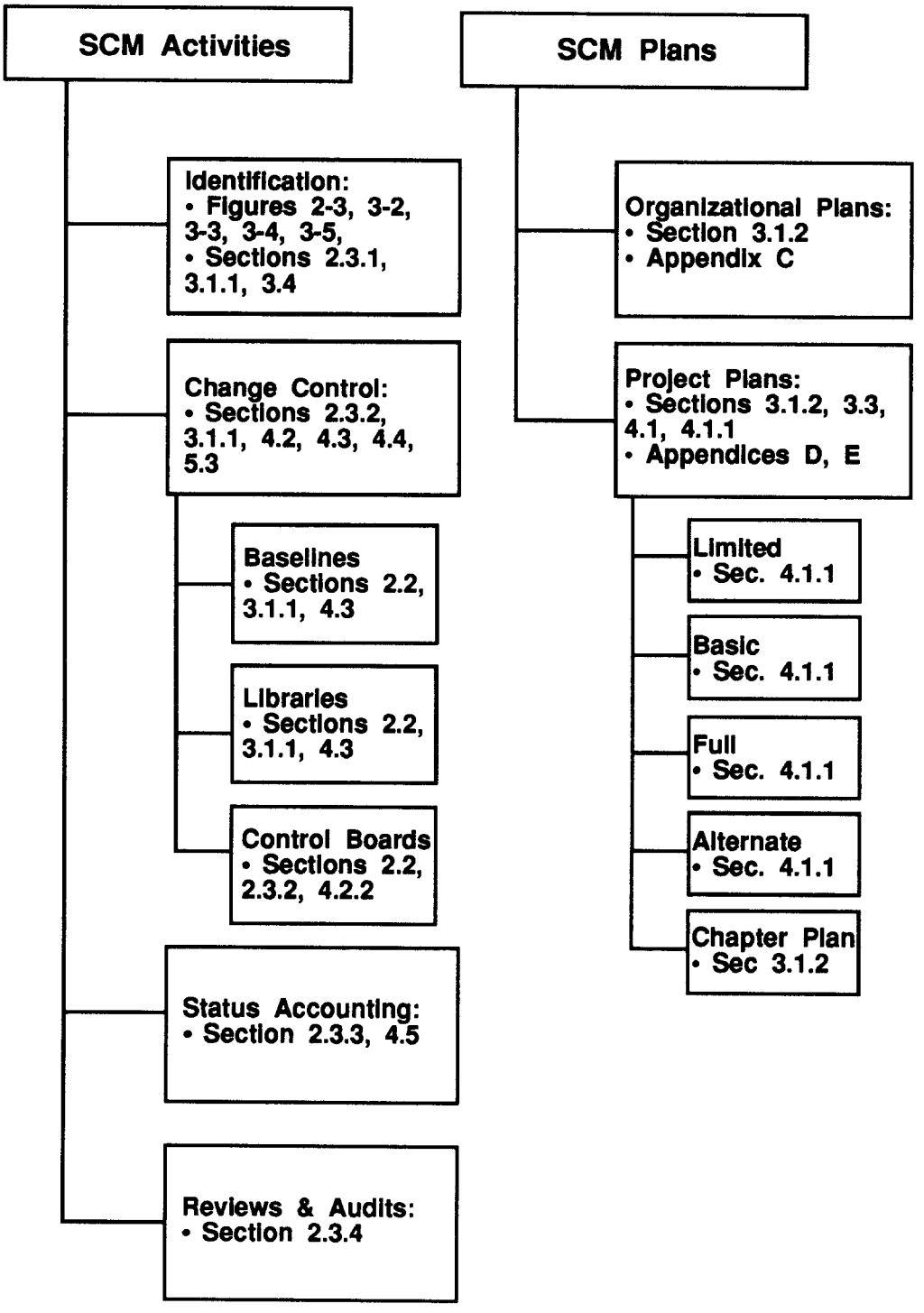


Figure 1-1. SCM Application Summary

2 Configuration Management Concepts and Activities

Configuration Management (CM) is defined as "the process of identifying and defining the configuration items in a system, controlling the release and change of these items throughout the system life cycle, recording and reporting the status of configuration items and change requests, and verifying the completeness and correctness of Configuration Items." [IEE90-a] In essence, CM is the collection of activities performed during a project to: (1) determine and identify those parts of the system that need to be controlled, (2) ensure those parts have necessary and accurate definition and documentation, (3) ensure changes are made to the parts in a controlled manner, and (4) be able to tell at any point in time the status of a part (*e.g.*, whether a specific part is completed, being changed, waiting to be tested, or perhaps released to the customer).

2.1 Objectives and Benefits of Configuration Management

The software life cycle process model of Figure 2-1 has a very linear flow, progressing logically from one phase to the next as the emphasis of activities of one phase decreases and the next phase increases. However, this "waterfall" life cycle model represents an ideal situation which does not occur in reality. In practice, the software life cycle is a highly iterative process.

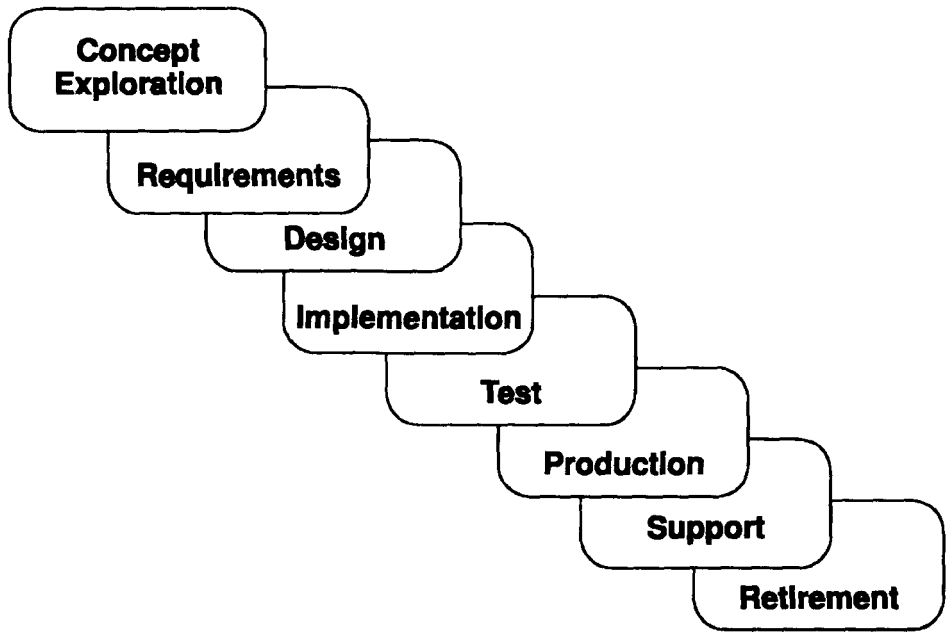


Figure 2-1. Software Life Cycle Process Activities

As shown in Figure 2-2, changes can, and do, occur at all phases in the development portion of the life cycle. Design changes may be necessary if requirements change, or when deficiencies are identified in the design approach to a stable requirement. (Of course, early negotiations with the user, customer, or "customer base," if multiple customers exist, can establish a more stable foundation at the beginning.) Implementation changes may be encountered for similar reasons. Testing may uncover defects that require changes in the code and/or the design and requirements.

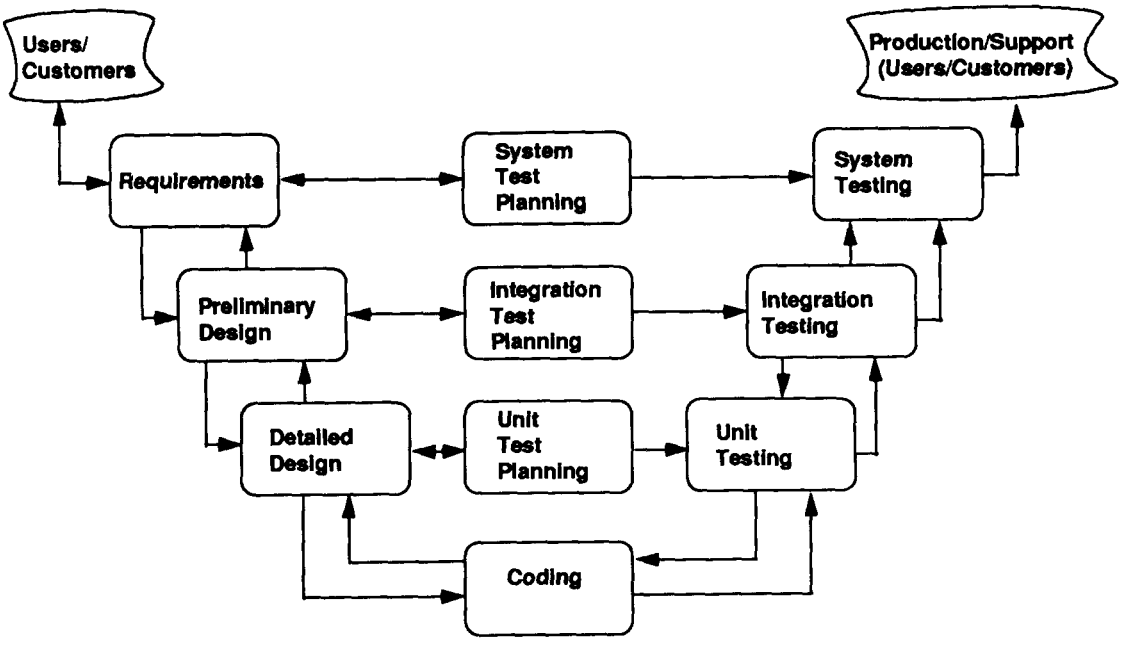


Figure 2-2. Relationship of Software Life Cycle Process Activities

While the development of a new software product is certainly a fluid process, so is the modification and support of a product which has already been released. Changes must be made to the right issue of the code, testing must verify performance of the change and the integrity of the remaining software, and all associated documentation must be made consistent with the final code. In essence, the support process undergoes a fluid development cycle of its own which parallels that of the original design process.

When change is such a part of the creation of the product, it is not hard to imagine that keeping track of the following issues can easily become a nightmare.

- Who has which version of a particular part of the software?
- Has a particular change been verified or tested?
- Which test results match which versions of the test plan and the code?

- What actions remain to be completed or which pieces of the software need to be finished for the project to be completed?

- Which changes will be included in the next version of the product and which will be deferred for later?

The Basic Reason for SCM

The basic reason for implementing a configuration management system encompassing all software products is to keep the constantly changing and iterative software components and the associated documentation in a non-degrading state throughout its life cycle. This is a challenge that must be met in order to develop and maintain quality software. The quality of the software is fundamental to the level of quality of the complete system.

SCM as a Management Tool

A principal benefit of software configuration management is the ability to give an instantaneous view of the most current state of dynamically changing software to management as well as to developers. This means configuration management of the software life cycle should allow decision makers, at anytime during the iterative process of development, to have an instantaneous picture of a software project from which to base decisions. As well as having system status available to management at all times, the status of any module during development can be determined. This allows the progress of all modules to be tracked independently of the others. Armed with this knowledge, management and developers alike are able to make better decisions about the future of that software project and more realistic resource forecasts for the development and support effort.

Concurrent Development of Code Modules

A software configuration management system allows for, and helps track, concurrent development of the different modules or components which make up the overall system. It can also prevent two or more people from making changes to the same module at the same time. This becomes more important as the size of the project increases (and the number of components, interfaces and interactions increase, or the size of the development team increases). This concurrent development process and simultaneous change protection allows for the overall progress to be faster. Perhaps more importantly, the overall development remains continuous because the SCM system can provide visibility of the entire software system to all developers. Even for small projects, the configuration management system plays an important role by keeping all the concurrently developing code and its associated documentation organized. This saves the project time in the end because configuration management has forced each phase of the software system development to be organized and executed in a documented, prescribed manner.

Code Reusability

If software engineering techniques and configuration management are rigorously adhered to, they provide a means of developing a library of reusable code. In such an environment, each module of code would have its purpose, function, and interface meticulously defined,

unambiguously documented, and thoroughly tested. This would allow other developers the opportunity to reuse one of the previously implemented modules with little or no modification.

2.2 Fundamental Concepts

Principal Terms

- Configuration Item - Reference Figure 2-3. A software Configuration Item (CI) is a collection of software entities (or components) treated as single element for the purpose of configuration management. Configuration Items can vary in size, complexity, and type. A CI may also be called a Computer Software Configuration Item (CSCI), Computer Program Configuration Item (CPCI), or (software) system segment. CI or CSCI is used in this guide. In a small to medium project, the entire collection of software requirements, design description, code, and tests may be associated together under one CI.

- Computer Software Component - Reference Figure 2-3. A Computer Software Component (CSC) is a distinct part of a software Configuration Item. CSCs may also be further decomposed into subordinate components or individual units. If a large analysis program is called out as a CI, some of the CSCs could be entities such as a requirements document or functional groupings of software modules. A CSC may be called a Computer Program Component (CPC). CSC, or just software component is the term used in this guide.

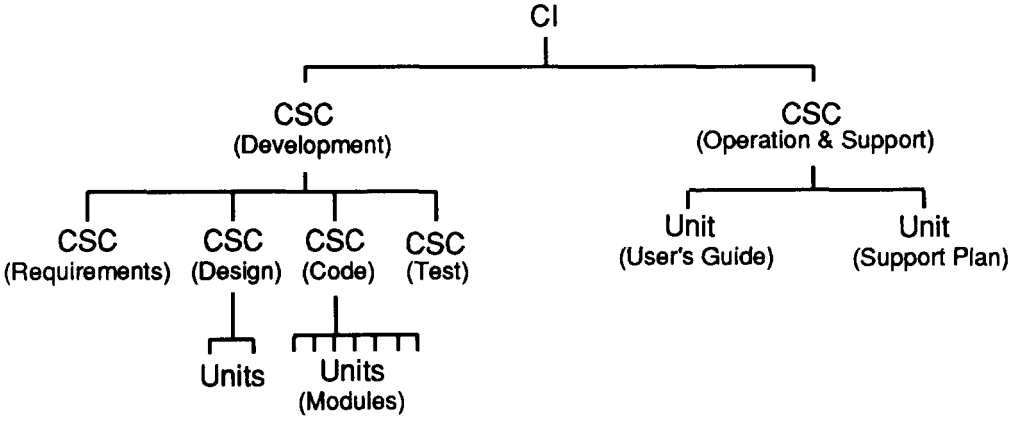


Figure 2-3. Sample Configuration Item Decomposition

- Version - A version is a software CI or CSC with a defined set of capabilities. A new version is a variation of the previous version, in that it has a change in its functionality or performance characteristics. An example of this is a change to the software to generate a different output.

- Release - A release is a copy of the software CI or CSC that is turned over to the customer or user. It is a promotion of that CI or CSC outside of the development environment.

- Revision - A revision is a formal change to a software CI or CSC that does not alter its documented functional or performance capabilities. An example of this is when code is changed to correct a fault.

- Baseline - A baseline is the documented identification of a software product CI — its code and all its related documentation (*e.g.*, CSCs) at some specific point in time. It is the basis for all SCM activities. When a baseline is released, it is usually called a new version. CSCs may be baselined at varying intervals in the development process. Baseline is the designation that signifies appropriate justification and approval are required to make changes to the baselined item.

- Personal Library - The Personal Library (also termed the dynamic or programmer's library) is used for holding newly created or modified software entities. This library constitutes a software developer's workspace for writing new code or documentation, and may take any form suitable to the developer's needs, but should have a degree of order to it that allows the status of its entities to be determined easily. Each software developer may have a Personal Library from which project entities can be linked and/or copied. Or, each small team may have a Personal Library assigned to the team with lower level personal libraries assigned to each individual, especially in a Local Area Network (LAN) or mainframe environment. Access to the Personal Library is controlled by, and usually limited to, the software developer.

- Project Library - The Project Library (also termed the controlled or "master" library) is a library used for managing the current internal developmental baselines and for controlling changes made to them. This library represents the latest internally-approved version of the software product being developed. Changes to software entities in the Project Library should have gone through formal approval procedures established in a configuration management plan. Code in this library should have been tested sufficiently to assure that it is ready for integration. Copies may be freely made for use by the software developers and others, but changes must be strictly controlled and documented in order to ascertain at any given point its exact configuration. Even for simple projects in which there is only one code developer, there still should be functional separation between the Personal Library and the Project Library.

- Release Library - The Release Library (also termed the static or "software repository" library) is a library used to archive the various baselines (versions) released for general use. This library is never changed (except to add a new version), since it must be able to duplicate results from software that has been released for operational use by other organizations. Access should be limited to "read only" for the purpose of making copies. The Sandia Drawing System provides a Release Library capability required for WR projects and is useful for most other projects involving formal software deliverables.

- Promotion - A promotion is an action taken with a software component to increase the level of authority needed to approve changes to it. For example, a top-level Software Design Description (SDD) is promoted or moved into the Project Library where all developers can view, but not modify it without proper authority. This allows the developers to work on issues that may concern detailed designs and implementation.

- Librarian - A person designated to exercise physical control over one or more configuration management library.

- Configuration Control Board (CCB) - A Configuration Control Board is a group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes. A CCB is sometimes referred to as a Change Control Board. CCBs may be responsible for the system as a whole, the hardware alone, or the software alone. If the board is only concerned with software it is usually called a Software Configuration Control Board (SCCB).

- Software Configuration Control Board (SCCB) - A Software Configuration Control Board is a group of individuals who oversee the software change process, with ultimate authority for approving a change and promoting a software entity from one library to another. The individuals may be from the project, related organizations and management levels, the customer, or some combination. During the development process, the SCCB controls promotions into the Project Library from the Personal Library and changes to the products in the Project Library. During the support phase, the project CCB and the SCCB provide the authority to make changes to products already promoted to the Release Library, and to promote software products from the Project Library to the Release Library. An SCCB may also be referred to as a Software Change Control Board.

Relationships between Libraries

During the development of the software and its associated documentation, work products can be kept in the designer's or programmer's personal workspace, referred to as part of his or her Personal Library. Note that more than one person may be working out of this Personal Library in some instances. As the work progresses, copies can be passed to other interested parties for criticism, feedback, or general information. When the component seems to have reached a stable point during development, it is approved and formally placed in, or *promoted* to, the Project Library as part of a baselined* CSCI. (At Sandia, depending upon the initial structure of the configuration management system that was decided on, this software component could be placed into the Drawing System even though it has only been promoted to the Project Library.) All project members have access to this library but cannot arbitrarily change the component. Additional changes may be made to this baseline. After approval of any additional changes, it may then be promoted into the Release Library as a new release (and, different suffix, denoting a new version, if different capabilities have been incorporated). From this Release Library a revision, or a new version, can be transferred to the customer or user.

* Baselines represent different hierarchical stages in the development of each software entity. For example, in the process of building a computer program, program units (e.g., procedures or modules) are developed. Each unit may be referred to as part of a code component baseline and each should be assigned a unique and documented identifier. During the development process, the component baseline becomes more and more complex as individual units are integrated with one another until eventually the baseline configuration represents the final software product. An important part of this process and a key aspect of configuration management plans is the promotion of baselines from one stage of development to the next or from one activity to the next (*e.g.*, from development to testing). The requirements for promotion and the authority responsible for promotion should be stated in the configuration management plan.

2.3 Specific Activities

2.3.1 Identification of Configuration Items

The first step in the configuration management process is the identification of the items to be managed and the specification of the management authority responsible for each item. This is a critical step in the process since the identification scheme is employed throughout the life cycle of the software product.

Examples of some of the software components which may be included in the configuration management identification include: design and requirements specifications, source code, executable code, support software, test plan, test data and results, data bases, software support documentation, user documentation, and even plans such as the Quality Assurance, Software Development, and Configuration Management Plans. (Subsets of these software components may be identified separately (CSC or unit) for inclusion in multiple applications.) The levels of authority assigned responsibility for each configuration item will vary depending on the complexity of the software product, the size of the development team, and the management style of the responsible organization.

An important aspect of configuration identification is the use of a formal naming convention for each entity. This naming convention, which typically utilizes a combination of mnemonic labels and version numbers, should be applied to all components of a Configuration Item. In establishing the naming convention, consideration should be given to system constraints such as name length or composition. Consistency in the application of this identification scheme is critical to accurate tracking of the software development process.

2.3.2 Change Control

The software development process involves a natural progression of change and improvement. Uncontrolled change, however, is often counterproductive and may result, at the very least, in wasted effort or missed milestones. For this reason, change control is a critically important feature of configuration management.

The use of software libraries is important to the successful control of software change. (See **Relationships between Libraries** in Section 2.2.) The two primary libraries for controlling changes are: personal libraries and project libraries. In the Personal Library, newly created or modified versions of software entities are maintained by the responsible developer. Periodically, work products may be promoted to the Project Library where access is granted to other developers or interested parties who require all or part of the promoted entity to be stable enough for their portion of the development effort. A formal method should be in place to process this promotion activity (*e.g.*, a "software inspection" of the work product).

A mechanism should be established to process change requests (*e.g.*, discrepancies, failure reports, requests for new features) from a variety of sources throughout the development process and during operation and support of the system. This mechanism should extend to a definition of a process to track, evaluate, and implement change requests into a new version and new release of the software. The following paragraphs highlight important activities in such a process.

Approval of change requests and subsequent change implementation to the Project Library should be handled by a configuration control authority or Software Configuration Control Board (SCCB). Depending on the size and nature of the software project, the SCCB may consist of a single Librarian, or there may be more than one SCCB, each with a different functional responsibility and jointly responsible for common interfaces, if they exist. The SCCB takes responsibility for ensuring that changes are implemented and tested according to standard procedures and that hardware/software interfaces and interfaces between software modules are not violated. The SCCB also focuses on overall project management responsibility for ensuring that design or requirements specifications are not violated and that software changes are implemented according to cost and schedule constraints. Additionally, the size and structure of SCCBs normally change over the life cycle of the software.

To simplify the task of the software configuration control board, formal procedures for an initial analysis of the change should be established to aid the board in prioritizing the change. Review and testing requirements for prospective changes should also be established. Results from reviewing and testing the change should then be submitted to the SCCB to assist in evaluating the change.

Formal procedures should be established for promotion of a baseline into the Release Library and for subsequent access to the baseline by the project staff or end users. These procedures should specify the format for baseline submission and the documentation that must accompany the submission. Once a baseline configuration is submitted to the Release Library, changes to the baseline should be strictly controlled.

The Release Library serves as a repository for each baseline, and allows an historical record to be kept of the baseline development and all associated documentation. This library also provides a central location for the receipt and processing of formal requests for change to a release and the subsequent tracking of the status of changes.

Only SCCB-approved changes to baselines residing in the Release Library should be allowed. Documentation that should be submitted with the approved change include: the associated change requests, updated design documents, reports on the review and testing of the change, information on implementation of the change, and identification of any supporting software that may be required for implementation.

At every library level, implementation of a change into that library produces a new baseline with a new and unique identifier. Also, at this time the software librarian should produce backup files and update system build procedures if necessary.

Control of changes to support other third-party software presents a special challenge to the configuration management plan. Such software entities are usually maintained at a different site from the primary software product and are usually not subject to the same configuration management plan. Still, controls should be in place to ensure the continued availability of all required support software including compilers, operating system, etc. It may be desirable, therefore, to retain support software along with the primary software product.

2.3.3 Status Accounting

The formal process of tracking software entities through the steps in their evolution is referred to as status accounting or *configuration status accounting* (CSA). Status accounting is an important part of software project management since it provides the project manager with the data necessary to gauge and report project progress.

Status accounting reports are most conveniently filed at logical transition points in the development process. For example, when a baseline configuration is advanced from the design emphasis to an implementation (coding) emphasis, a record of this should be filed in the configuration records. Similarly, when a newly developed or modified module has been approved for system testing and integration, this transition in status should be recorded.

Status accounting encompasses more than just a few reports. The purpose of status accounting is to have the ability at any point in the development process, to provide the current content of a given software CI or CSC. If the entity is a software design description, a status accounting mechanism should allow developers to easily pull together any text files, graphic files, and data files that may comprise the document. Likewise, change requests to software products should be trackable and the status of these requests should be readily producible in an organized manner.

2.3.4 Audits and Reviews

Audits and reviews are performed to verify that the software product matches the capabilities defined in the specifications or other contractual documentation and that the performance of the product fulfills the requirements of the user or customer.

There are two types of audits which should be performed prior to release of a product baseline or a revision of an existing baseline. A *physical configuration* audit should be performed to determine whether all software items which should be part of the product baseline are present in the version specified in the current status report. A *functional configuration* audit should also be conducted to ensure that the software product satisfies the functions defined in the specifications. An important aspect of these two audits is the assurance that all documentation (change requests, test data, and reports, etc.) relevant to the release are current and complete.

In large software projects, audits may be necessary throughout the evolution of a product to ensure conformance with planned CM actions and to prevent massive problems from being encountered just prior to release.

A review of the configuration management plan should be periodically performed to assess the effectiveness of the approach and the extent to which configuration management procedures are being followed by project staff. This allows adjustment of procedures to improve the staff's ability to follow a set of procedures and to allow for more effective approaches to be incorporated as they are developed.

2.4 The Software Release Process

The culmination of any software development effort is, of course, the release of the software to users. The two steps in the release process are the preparation of formal backups and the verification that all necessary documentation has been completed and matches the code approved for release.

Backups should be prepared for all files that are needed to rebuild and test the software. Examples of backups which should be prepared include: all source files, build procedure files, data files, and executable files. In addition, instructions should be prepared describing reinstallation and testing of the system from the backups.

The documentation included in the release package should describe to the user the features included in the release and how to install and use the software code (and its supporting documentation). The completeness of the documentation will vary depending on the size and complexity of the release or the criticality of its application. Contractual obligations may also dictate the type of documentation which must be included in the release.

The package of information associated with the release is sometimes referred to as the *Version Description Document*. Some of the information which may be included in the Version Description Document include the following:

1. A list of the contents of each tape or diskette provided with the release. A brief description of the contents may also be provided.

2. A functional description of the release to inform the user of any new capabilities not included in previous releases.

3. A list of previously identified problems which have been corrected in the release. Specific problem reports that have been resolved may be referenced here.

4. A discussion of special user considerations such as actions required to use the new release (perhaps update pages to a User's Manual). Limitations or potential problems with the new release may also be discussed along with any plans to correct these limitations or problems in future releases. This information is provided to prevent the user from filing duplicate problem reports or change requests, and to prevent potential disasters to the user's applications.

5. An inventory of the source, executable, and data units included in the release. The inventory is usually listed in alphabetical order with a designation of the version number and a creation or modification date.

6. Special instructions, if any, for installing the software on user computing systems. If this information is provided, a walkthrough should be performed to ensure that the complete set of instructions required for installation is provided to the user.

If the release is a new software product (i.e., not a revision of existing software) or a major revision of existing software, it may be necessary to provide more extensive documentation than that included in the Version Description Document. This additional documentation, which is sometimes referred to as a user's manual or a code reference manual, may describe in much greater detail the structure of the code and the individual program units. Finally, in some applications, such as computer modeling of physical phenomena, the theory underlying the actual source code may be described in a separate document referred to as a code theory manual.

2.5 Integrating SCM Activities with the Software Life Cycle

When to Start

The time to start considering a software configuration management system is at the beginning of the project.

The SCM system should have in place a change authority or Software Change Control Board which can review the impact and technical consequences of any change on the design, code and testing already undertaken. If changes are approved and incorporated, the Project Librarian tracks and records these changes to the system to allow for traceability of change process. Any of the products of the software engineering process may go through several changes, and the software configuration management system needs to ensure that all those working on the system affected by the changes are aware of their presence.

If a change to the software turns out to be a mistake, the SCM system should allow an immediate return to the state of the software and its associated documentation prior to that change. The flexibility to do this allows many possible avenues to the solution to be experimented with in detail, which makes it valuable during rapid prototyping design processes. It also tends to significantly reduce the effects and recovery time of large mistakes.

Concept Exploration Phase

During this phase, the specifics about the SCM system should be decided. A Librarian for the project should be selected at this time. The Librarian has the responsibility to physically establish, control, and maintain the collection of software components that constitute one or more baselines of a software system. Depending on the size of the project, the responsibility of the Librarian can be substantial. Standards should be set regarding the different ways of communicating information about the software project; when can it be given, and what can be conveyed to whom? How formally will each transfer be handled? What is the protocol and how is it recorded? Several of these decisions depend on factors such as: who the customer is, how big the project is, and whether personnel from more than one organization are working on it. Positions of responsibility should be assigned to those who will be involved in the process.

Requirements Phase

A formal record should be documented (a Software Configuration Management Plan or SCMP) and maintained which explicitly states all the decisions that were made about configuration

management in the Concept Exploration Phase. It should include decisions like: librarian's responsibilities; libraries that will be used and their formal definition; numbering of changes.

By the time focus is shifting from requirements definition to design, the configuration management system should be firmly in place for the rest of the life cycle. This will allow for changes made in later phases that affect requirements to be tracked and managed effectively. The net result is a requirements specification that remains current through development and is a viable asset during operation and support.

Design Phase

The software configuration management system will allow multiple variations (if necessary) of the design to be tracked and managed effectively. As a design evolves, its history is recorded in the configuration management system so that a previous state can be returned to, if necessary. The history will also illustrate why certain design directions were abandoned and others taken. This can save a great deal of time if the staffing on the project is dynamic, since the "corporate knowledge" as captured in the SCM system is readily available.

Implementation Phase

Software configuration management throughout the life cycle allows concurrent module development (as well as multiple variations of the same module to be developed if a goal of the project is to explore implementation alternatives). Management can assign several programmers to decrease implementation time because of the ability to develop the modules simultaneously. An individual programmer will also have the flexibility to try different approaches to the development of that module without affecting the overall implementation.

Integration and Testing Phases

During integration, the modules that make up a particular version or revision of the system should be easily identified, even after multiple iterations through the previous phases of the life cycle. Also, the testing requirements and their associated tests and documentation, which have probably also undergone various changes during the life cycle, should still be linked correctly to each of the prospective modules as well as the version or revision they produce. If a satisfactory system cannot be built, then the history of each particular module and its associated documentation can be traced back into its development in an effort to locate and isolate the defect.

Installation and Checkout (Production) Phase

This can be an intensive time frame even though it may be very short. For those projects that perform a separate and distinct installation on a user's platform, the ability to rapidly access all or portions of the current configuration is essential. If a new problem suddenly arises, as it likely will, the ability to rapidly identify and isolate the problem area, recommend and approve the change, and perform testing, relies on the configuration management system being able to respond as in integration and test, but it must also function in this higher level stress environment.

Operation and Support Phase

During the operation and support phase the configuration management system will again have the tools necessary for a quality product. Product definition documentation that has been generated and modified all through the development life cycle can be extracted from the configuration management system and packaged independently for the user. This process should be very straightforward if the documentation for the associated version or revision is already written and in place.

Another asset the configuration management system will offer is change tracking. Even though it appears the development effort is over, the system will continue to track any changes made during the support phase. Software support personnel need to know how those changes affect the total system: software, hardware, and all the associated documentation. Each change should be formally and carefully monitored by the Software Configuration Control Board before its inclusion into the system.

Probably the most valuable use for the configuration management system's now formed database will be to provide all the information new project personnel need to learn and modify the system. The configuration management system has all the modules and their history linked and available in one place. This system will allow new personnel to come up to speed on the complete system much more quickly.

Retirement Phase

The SCM system should be kept active until the software is retired. The final benefit the software configuration management system will offer is that it should contain any specific requirements for a safe and secure retirement of the software system. When the system is retired in full, the configuration management system is used to identify released software that must be recalled or replaced by other software. Some records may be archived with the retired software but the SCM system (the process) can certainly live on, providing the tools to manage the next project.

3 Planning for Configuration Management

Planning for software configuration management (SCM) requires some forethought on the part of the software developer. Even though many of the activities associated with SCM are routine, their interaction as they occur throughout the life cycle of the software product requires definition. The success of a software product depends on (1) clearly stating the actions to be performed to provide visibility into the evolving configuration, (2) providing the mechanism to implement changes before and after software components are formally baselined, and (3) assuring that changes have been incorporated into the appropriate computer software components.

Software configuration management planning is identified as a part of planning for software quality. As shown in Figure 3-1, the Software Configuration Management Plan (SCMP) addresses in more depth, several areas identified in the Software Quality Assurance Plan (SQAP) [IEE89]. The sections of the SQAP that are bolded in Figure 3-1 are areas that an SCMP would amplify as the circumstances warranted.

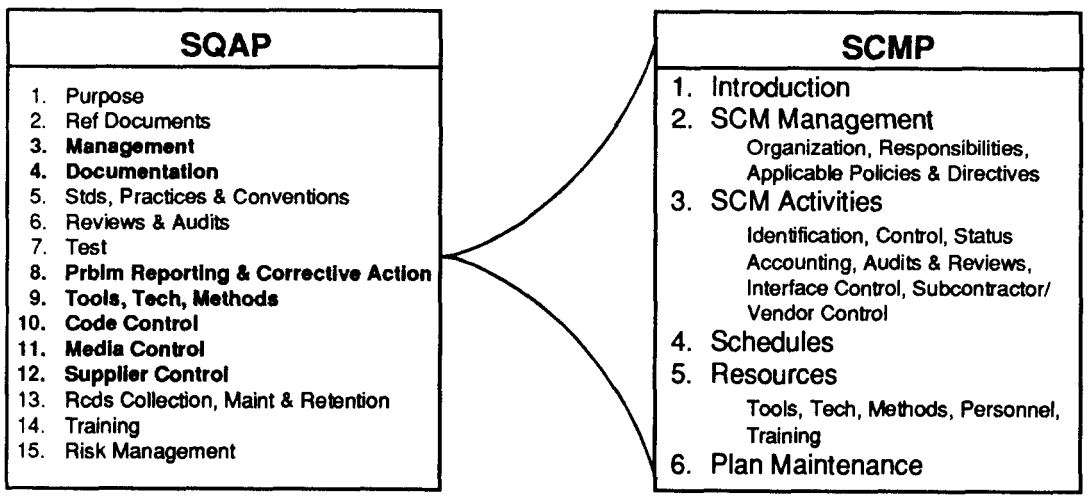


Figure 3-1. SQAP - SCMP Relationship

3.1 Software Configuration Management Plans

There are six elements, or sections, to an SCMP. Section 1 contains an overview of the plan itself; the software, life cycle phases, and activities covered in the plan; definitions; and references. Section 2 addresses organizational structure and associated responsibilities, and governing policies and directives. Section 3 describes how SCM requirements will be satisfied through identification and control of software components, status accounting of software components and changes to those components, audit and review functions, interface control, and subcontractor and vendor software.

Section 4 describes the implementation schedule of SCM activities while Section 5 addresses resources (tools, methodologies, personnel, and training.) Section 6 discusses maintenance of the plan itself.

Planning for configuration management should be done early in the project development process as discussed in Chapter 2. This does not necessarily mean that a large, formal SCMP must be developed for every project. The SCMP for a project may be a section in the Software Quality Assurance Plan (SQAP) or the Software Project Management Plan (SPMP), sometimes referred to as the Software Development Plan (SDP). This is especially true if there exists an organizational SCMP describing the way configuration management is normally performed for all software developed by that organization. The following sections will bring into perspective the level of planning that may be applied to a software project.

3.1.1 Essential Elements of an SCMP

Referring again to Figure 3-1, every plan usually has a fair amount of "boilerplate" associated with it and an SCMP is no different. This consists of the purpose, scope, references, and definitions of Section 1, and to some extent, the organizational structure, responsibilities, and applicable policies, directives, and procedures of Section 2. While these are not to be discounted, especially for larger projects, the following discussion is centered on the particular elements that are unique to configuration management. These are Sections 3 through 5 of the SCMP. Section 6, Plan Maintenance, is not addressed here (see [IEE90-b] for more information).

What are the components to be developed that require Configuration Management?

One of the first things to be done is to identify the software components and CIs that are going to be developed for a particular project. Part of the identification process is matching a controlling identification scheme to follow the structure of the software being managed, which in turn may be dependent on the application. Within the software product itself, there must be an identification method to relate the various human-readable components and the binary (non-readable) components. Human-readable components include the Software Requirements Specification (SRS), SDD, the source code in hard copy, Test Plan, and some automated test scripts. Non-readable components include object code, executable code, and compilers, linkers, etc. Identification should also include a module/table naming convention and revision identification scheme.

Where will the "official" version(s) reside?

An "official" version of any software component is one that has been baselined. Baselining* is a decision process that fixes a component in a particular configuration and must always encompass inspection or detailed review of that component. Basically there are two forms in which software components reside: magnetic media and hardcopy (or film). Both of these forms can constitute the three "Libraries". These three libraries, Personal, Project, and Release Library (see Section 2.2), facilitate conceptually, as well as physically, the amount of control being placed on the software components.

* The process of reviewing (inspecting) a component, agreeing on its contents and documenting that contents, and the agreement activity. See Principal Terms in Section 2.2.

Libraries for baselined components are the Project and Release Libraries. The Project Library may be on the same physical system as the Personal Library where the component was located before inspection or review. However, they should reside on different disks (directories) or tapes. Additionally, whether the media is hardcopy or magnetic, the Project Library should be under stricter control than the Personal Library because more than one person usually needs access to the Project Library. This includes protection against write/delete privileges for all but a designated configuration manager or Librarian.

The physical location of the Release Library should be separate from the Personal and Project Libraries. This is really a more permanent record of the last baseline in the Project Library. On the same system (disk or tape), they could be confused, even after changes are made in the Project Library, as work progresses toward the next release. The magnetic media format for all but executable code should be a project-common editor for code and a common word processing format or American Standard Code for Information Interchange (ASCII) for documents. (For archive purposes, ASCII is a safer format since word processing standards change over time.)

When will each software component be baselined?

The timing of when to baseline each component (Software Requirements, Software Design, Test Plan, Source Code) depends on the development approach for the project as a whole. Development projects that undergo a series of two or three prototypes will be more convoluted than a project that adheres to a very linear approach. Rapid prototyping presents additional challenges to ensure requirements and design information are well documented and baselined before the source code is finalized. However, regardless of the development approach, each software component that provides the basis for succeeding development work and a succeeding component, should be baselined before serious consideration can be given as to the validity of the succeeding component.

The key element in these types of situations is to make every effort to baseline the Software Requirements Specifications at the earliest opportunity. This may not be possible until the initial prototyping effort is nearly complete. It may be only then that the requirements are well understood. **However, further development should not proceed until these requirements are established and defined.** Only with the establishment of a requirements specification are the software design document and test plans in a position to be fully developed to match those requirements. Also, the source code should be baselined prior to integration and system testing. As with the other baselined software components, changes are then made within the structure of a configuration management scheme. This is essential during the final testing phases to ensure that changes do not adversely impact the validity of previous tests, and if new tests are required, there must be assurances that changed software is covered by the new tests.

Not mentioned specifically in the above discussion are the planning documents necessary for any software project. Without solid planning, there is little hope for real control over a development project. *If you don't know where you are going, any road (usually the long one) will get you there.* A Software Project/Development Plan, Software Configuration Management Plan, Verification and Validation Plan (if required), and organizational plans such as a Software Quality Plan should really be established and baselined at the earliest phase of any software development program. Early establishment of the development approach, methods and timing of SCM, and other

methods to ensure a high-quality and reliable software product provide the solid framework necessary to efficiently develop quality software.

How will changes be handled?

Software was meant to be changed. But changes must be accomplished in an orderly and progressively formal manner. That is the *raison d'être* for the three libraries previously discussed. Each library provides for both an orderly change process and a more controlled and increasingly formal structure than the library preceding it.

While the software component is still in the developer's Personal Library, that individual is solely responsible for making changes. The developer should use self-discipline to ensure changes are orderly but there is usually no prescribed formality, nor should there be. When a particular software component has been baselined through an inspection or review process, it should be promoted to the Project Library. As mentioned before, the physical system where this library resides does not have to be different from the Personal Library. The Project Library must be a different directory, however, preferably on a different disk/tape* . It can be copied into any Personal Library for further development work. Note that each time the baseline in the Project Library is updated with one or more changes, the Personal Library must reflect that update. Each iteration of the Personal and Project Libraries should be saved for traceability. As an alternative to saving the entire component, the differences (*e.g.*, a "diff list") can be saved.

For changes to be incorporated into the Project Library, they must be introduced through the configuration management process under some change authority. For this level of configuration management, changes should be at least peer reviewed. (The software inspection is the recommended review method [SSGv3].) As the number of people working on a project diminish, this becomes more difficult. One-person projects present the most challenge because there may not be enough individuals with a working knowledge of the project. This requires dedication on the part of those individuals to at least "walk through" the change(s) with a co-worker or software-knowledgeable member of the host-system development team. Each successive component (Specification, Design, etc.) requires more insight into the details of the software than the preceding component. Organizations producing software that typically dedicate one person to a project would serve themselves well to establish a standing group of developers to draw upon for this, review process.

The highest level of control and the change process requiring the most formal methods of configuration management is that of software contained in the Release Library. Once a baselined component is promoted to the Release Library, changes should only be implemented with a new release through a change authority such as a Software Configuration Control Board (SCCB). The size and personnel mix of the "board" will depend on the software being developed. The SCCB does not approve changes through strength of numbers but through a dedicated, thorough examination/inspection of the change(s), including their impact. If changes include changed code, test results using the new code should be included in the examination.

* Project Libraries themselves should be in a central location and in a standard format for that project under the control of the Project Librarian.

Layered over the physical changes to entities in libraries is a change-tracking mechanism. This tracking provides change history information and allows management of the "change request" or "discrepancy notice" activity. This mechanism is more formally controlled as software components are promoted from one library to the next.

3.1.2 Types of Plans

A Chapter in Another Document

As mentioned before, every plan has a fair amount of boilerplate. For most of the plans associated with a project, much of this is the same. Overviews, system descriptions, responsibilities, interfaces, schedules, and references change very little, if at all. For projects with one or two people as the responsible developers, the usual practice is not to generate plans but to begin work on the project itself. This should not be viewed as an excuse not to plan, but the reason to plan smartly.

Discussing software configuration management as a section within a Software Quality Assurance Plan, Software Quality Plan, or Software Development Plan capitalizes on the structure and information already present in the "host" plan. However, confining SCM to a chapter within another document is not always the best answer. To use the chapter approach, the size of the project should be small and the organizational interfaces need to be small, and be within Sandia. There should not be a continuous software support responsibility. Of course, the assumption is that the customer is not requiring a separate Software Configuration Management Plan.

SCM as a chapter is an enumeration of the essentials discussed previously: describing the scheme to identify and document the functional and physical characteristics of the configuration item(s) (CIs), describing the methods to control changes during the stages of development, and describing the methods to track the status of the CIs.

A Stand-alone SCMP

Some software development projects are sufficiently large and complex that two or three programmatic documents are necessary to adequately describe the activities, interactions, and infrastructure of the project. The SCM activities may warrant separate treatment in a stand-alone SCMP. Situations that can lead to this are: partitioning of the software among diverse groups (at different sites); an on-going long term support function associated with the project; separate support programs, system exercisers, etc., must be developed and maintained; and/or the customer requests it.

The areas that should be discussed in a stand-alone SCMP are:

- Introduction/Overview - Brief introduction about the system, purpose and scope of plan, definitions, references.

- Management/SCM Responsibilities - Who the players are and what authority and responsibility they have.

- Activities - Describe how the players will accomplish their responsibilities of: identifying CIs, controlling changes, tracking configurations, reviewing (auditing), controlling interfaces and controlling subcontractor or vendor software.

- Schedules - Describe the sequence and dependencies of SCM activities.

- Resources - Discuss any tools or methodologies that will be applied as well as the use of Libraries and other control techniques. Describe the personnel resources needed and any training requirements to accomplish SCM

- Plan Maintenance - Who the owner of the plan is. Describe how changes to the plan itself are instituted.

Appendix F contains a template for a stand-alone SCMP.

An Organizational Software Configuration Management Plan

The organizational SCMP is written in more general terms. The goal here is to provide a standard for those software configuration management issues that are continually applied to all projects within a department, center, directorate or even a vice presidency. Even if projects differ widely, the existence of an organizational SCMP can allow project plans to draw on the organizational policies and procedures, thereby becoming smaller in size and easier to develop.

Appendix C contains a thematic for an organizational software configuration management plan.

A War Reserve (WR) Specific Plan

An SCMP for WR or WR-related software can be a chapter or stand-alone plan. However, there are some considerations that must be addressed:

> For Sandia, the WR or WR-related identification issue is governed by EP401045. (See Section 3.4, Using Procedures Effectively.)
>
> For a WR or WR-related system, the controlling identification scheme must include a six-digit part number with a three-digit suffix (revision number) so the host hardware system or next assembly will have a compatible, traceable link to this software component (or entity) just as to the hardware components. The six-digit part number is a top-level number or Materials List (ML) number.
>
> For documents, the identification normally is a two-letter prefix (*e.g.*, SR, TK) attached to the six-digit part number and the component name (*e.g.*, Software Requirements Specification, Software Test Plan). For file-based components, the identifying scheme is similar: a two-letter prefix (e.g., AT, AM) attached to the six-digit part number.
>
> When a WR or WR-related component is promoted to a Project Library it is essential that a Drawing System six-digit part number be obtained for it. An Issue 0 or Checkprint ML can be generated with the software components (drawings) identified This action provides two

important services. First, it serves as a linking identifier to other software entities associated with the software product. Second, and perhaps more importantly, the software component(s) can now be formally identified within the Drawing System as a part to a next assembly or to the system itself.

The Release Library for WR and WR-related software parts is the Sandia Drawing System. Software components must be in the Drawing System when a software part is formally released as part of a WR product or into service as part of a WR-related product. (Note that commercial software must be identified but reproduction from the Release Library may be prohibited due to copyright laws, even if it is needed to support a change of "Released" software.)

Most WR and WR-related projects are developed in a series of "builds" (e.g., Group 1 (breadboard), . . , Group 3 (advanced development), Pilot Production Item, Qualification Evaluation (Tool Made Sample, Qualification Sample)). The transition from one build to another should be heralded by baselining the software components as one or more configuration items.

Appendix D contains a stand-alone plan for a fictitious project using an embedded WR product.

A Research Project Plan

Like the WR project, a research project plan may also be a chapter plan or a stand-alone document. There is greater freedom in selecting a controlling identification scheme for non-WR applications such as a research project. However, these applications are not prohibited from using the part numbering scheme and the Drawing System like WR and WR-related projects. Since they are generally not as tightly integrated into the host computer as embedded systems, and if the Drawing System is not used, any alpha-numeric scheme that the developing organization consistently applies is acceptable. Part of the scheme must include a project identification sequence so that document and file based components have a common link. For example, all components of a weapons effects simulation could have WE1089 as part of its sequence, the "1089" representing the month and year the project was initiated. Another alternative is to use the acronyms of the title documents with a control number (part number) based on the project and an alpha-numeric version suffix. For example, an SRS and SDD could be identified as SRS89050-00 and SDD89050-01 (a revision of the original reflecting a change in the design to meet the original requirements). Most of these applications are larger than embedded systems and are usually written in a higher order language (HOL). They require more attention to naming conventions for modules, subroutines, segments, and variables mainly because there is more to manage.

Software in this category may also be "mastered" and stored within the Drawing System, Scientific Computing's Network Storage Service* (NSS) (requires a drawing system part number), or within the developing organization's designated repository, preferably in magnetic media **and** hardcopy for all components except compiled code.

Appendix E shows a sample stand-alone plan (and specific procedures) for a research project.

* Network Storage Service has replaced the Integrated Files Store (IFS)

A Plan for Reimbursable Projects

A reimbursable project may have requirements to rigidly adhere to IEEE 828-1990 or the DoD Software Development Standard, DoD-STD-2167a, as it addresses software configuration management as well as other areas. In lieu of external requirements it is highly recommended that EP401045 be used for identification and the Drawing System used as the Release Library.

Appendix F contains a plan template that would be useful in creating a stand-alone plan. Selected sections could be consolidated into a chapter plan.

3.2 Standards and Guides

There are several formal standards that a designer could draw from to develop an SCMP. These include ANSI/IEEE Standards and DoD Standards. DoD Standards are very detailed and require an extensive identification and tracking system. The principles are embodied in the ANSI/IEEE Standards and only these are presented here [IEE87, IEE90-a, IEE90-b]. In addition, a few books are available that discuss configuration management philosophy and application [EVA83, EVA87, BER80]. Also, *Sandia Software Guidelines* Vols 1, 3, and 5 [SSGv1, v3, and v5] contain related information that can bring configuration management more into focus as an important tool of the software engineering process.

3.3 Project Plan vs. Organization Plan

For many organizations an SCMP at the division or department level would be the most productive. If all or most of the projects are of similar application, similar size (in terms of code and manpower), and complexity (in terms of algorithms or functionality), the configuration management of their software products should be treated in a similar fashion. This allows minor differences to be treated in a smaller project document and referencing the organization document for most of the activities. This same approach should be taken for a Software Quality Assurance Plan; use an organizational plan to discuss the normal approach to software quality assurance and note any deviations, actual interface groups, and project specific methodologies or tools in the project plan. There are several areas or activities that are best suited to be managed at the organizational level.

Configuration Identification

The organizational plan should state what kind of identification scheme is used for different categories of software products. In most cases, the organization will be involved in either WR and WR-related, or non-WR such as energy research. Planning should center on the use of the Drawing System if WR, or an organizational identification scheme if not. In some cases where all types of work are performed, the discussion must differentiate between the two or state that the Drawing System will be used for all configuration identification.

Baseline Timing

Discussion of when software components are baselined can center on the activities that must be performed and agreements that must be reached before a specific component can be baselined. This

discussion should include the development approach normally used in the organization, e.g., whether there is a prototyping approach or series of "builds" (Group 1, Group 2, etc.) or a more singular approach. Specific timing and dates for a particular project are, of course, left to a project development plan with references to the organizational plan for the mechanics of the process.

Change Control

An organizational plan can describe the levels of control that organizations should be using to ensure that orderly and traceable changes are being made to software components. The discussion will differ depending on the Library (Personal, Project, or Release) in which the software component is currently located. The critical issues include how the organization wants to treat changes once a baseline is promoted to the Project Library and the Release Library. The discussion should describe the methods used to actually control the entry of the changes and will differ according to the size of the project and number of people who normally work on projects in that organization, *e.g.*, peer review, software configuration control (or review) boards. When control or review boards are discussed, the naming of the members should be left to the project development plan.

An organizational plan should also describe the basis for promoting software from one library to the next. These are the processes by which organizations manage software component development to ensure that orderly changes are made and baselines are properly established. Most of these processes are the same ones that provide for the verification of software components whenever verification and validation are discussed. These processes include software inspections, structured walk-throughs, peer reviews, and control (review) board activities and responsibilities.

3.4 Using Procedures Effectively

Sandia has an institutionalized approach that defines procedures for WR and WR-related product development. This system continues to modernize and has the necessary capabilities to document and provide an audit trail of the evolution of software components (Non-WR as well as WR and WR-related) as they are released for inclusion in the complete system. The elements of this system are defined in Engineering Procedure (EP) EP401040. Additional EPs describe part numbering and revision numbering (EP401016, EP401033, EP401054), and Computer Software Configuration Items (EP401040). As the system continues to improve, the ease with which it will function as a viable Release Library will also improve. For this reason, EP401045 is the recommended foundation for identification of all software developed at Sandia and the Drawing System is the recommended Release Library. Whether or not the Drawing System is used (particularly by non-WR projects), any organizational or project plan will be well served to use the concepts of EP401045 as the foundation of an independent approach.

EP401040 - Drawing System. EP401040 defines the Drawing System. The Drawing System allows for a two alpha-character prefix affixed to a six-digit part number, plus a three-digit *correlation suffix*, to identify the specific software drawing (computer software component). The six-digit part number and correlation suffix are also referred to as a *control number*. With the alpha-character prefix, the combination is referred to as a *support drawing*. Hence a support drawing to a software part number is a CSC.

EP401045 - This engineering procedure employs the concept of a Computer Software Configuration Item (CSCI) for identification of software components. Each CSCI is identified by a unique part number. Below are the codes for various components (CSCs) that could make up a CSCI for a software project:

SQ - Software Quality Assurance Plan (may also contain Software Quality Plan, Software Configuration Management Plan, Software Project Management Plan or Software Development Plan information)

SR - Software Requirements (Software Requirements Specification)

GR - Graphical Requirements (*e.g.*, Information Models, Data Models, Transformation Models)

SD - Software Documentation (Software Design Description)

GD - Graphical Design (*e.g.*, Data Flow Diagrams, State Transition Diagrams, Structure Charts, Flow Charts)

PD - Program Documentation (Source Code in hardcopy)

TK - Test Plan (Includes test cases and test results)

AM - Control Program (Source Code on magnetic media)

AT - Executable Program (The executable code which has been translated into relocatable or absolute machine code from the AM drawing)

AI - User Instructions (User's Guides, Checklists, etc.)

MP - Maintenance Procedures (Support program analysis, use, maintenance)

- No Prefix. The software part number or CSCI identification. A Materials List (ML) document will be used to list all required drawings, maintenance equipment, and support software. The software is assigned a unique six-digit number (initial part number suffix -00 or -000 for acceptance equipment) and is called out on the top drawing of the product which utilizes the software.

Figures 3-2 through 3-5 are MLs that show the use of the Drawing System to identify a software product and its supporting product definition (documentation). Figure 3-2 is the first release of the software product's components into the Drawing System. Changes to a component that create a new baseline may be reflected in an advancement of the "Issue" (*e.g.*, from Issue A to Issue B). This does not show up on the ML but would be apparent on the software component documentation. This could happen if changes in the Software Requirements were made to clarify an issue, defects were removed from design after inspection, or corrections were made to code in response to a test failure.

Figure 3-3 illustrates a significant change in the baseline of some of the software components, reflected in a change to the suffix. This represents a different version of those components with different capabilities and a new version for the software part itself (Issue B on the ML). This could have resulted from a new implementation of the design.

Figure 3-4 is an ML of a new version (different capabilities) of the software product (which is also a new baseline) that is backward compatible (*e.g.*, the new version can be used in applications calling for the original version; however, the original cannot be used in applications calling for the new version). This is shown as a two-digit suffix change to the software part itself. Note the issue of the ML is now C.

Figure 3-5 shows a new version that is not compatible with previous versions and hence, has a new product number. As an ML of a new part number, the issue of the ML is Issue A. Note that in this case, the new version is based on a revision of the original requirements.

SC----U-N-C-L-A-S-S-I-F-I-E-D----DRAWING NUMBER---123456 ISSUE---A
TITLE---SOFTWARE, MC1234 PROGRAMMER DESIGN AGENCY CODE IDENT---14213
TIE---SC/ / DRAWING LOCATION---SC DATE---02/17/91
TITLE CLASSIFICATION: UNCLASSIFIED----SHEET--- 1 OF 1

FOR EXPLANATION OF CODES SEE END OF CALLOUTS SECTION.

SMITH, J. 2833/TRAUTH, S. 7324/ 00100
00900
*** STANDARD NOTES *** 00910
*** OTHER REQUIREMENTS *** 1. STD NOTE D. QE REQUIRED. 00920

9900000 GENERAL REQUIREMENTS AND DRAWING INTERPRETATION 00930
9919100 MARKING, GENERAL METHODS 00949

00950
PRODUCT CHANGE HISTORY: 00960
00961

UNIT OF MEASURE / /	PRODUCTION AGENCY NUMBER /PENTAGON M/	CODE IDENT	DESIGN AGENCY PART NUMBER		PART CLASS /	I T E -M-	LINE NO
-00			123456-00		N		02000
--QTY--			--PART OR CONTROL NO	-----DESCRIPTION----			
NA			SR123456-000	SOFTWARE REQUIREMENTS, MC1234			02600
NA			SD123456-000	SOFTWARE DOCUMENTATION, MC1234			02610
NA			PD123456-000	PROGRAM DOCUMENTATION, MC1234			02620
NA			AT123456-000	EXECUTABLE PROGRAM, MC1234			02630
NA			TK123456-000	SOFTWARE TEST PLAN/RESULTS, MC1234			02640
NA			AM123456-000	CONTROL PROGRAM, MC1234			02650

----END OF CALLOUTS SECTION----

EXPLANATION OF CODES--
QUANTITY CODES--- AR-AS REQUIRED ARS-AS REQUIRED PER ASSEMBLY NA-DOCUMENTS ALT-ALTERNATE ITEM PM-PROCESS MATERIAL
EM-EXPENSE MATERIAL IALT-INSPECTION ALTERNATE
UNIT OF MEASURE CODES--- IN-INCHES LB-POUNDS OZ-OUNCES AVOIRDUPOIS TZ-OUNCES TROY G-GRAMS FT-FEET YD-YARDS PC-PER CENT
MM-MILLIMETER KG-KILOGRAM
PART CLASSIFICATION CODES--- UNCLASSIFIED: N

SC----U-N-C-L-A-S-S-I-F-I-E-D----DRAWING NUMBER---123456 ISSUE--- A

Figure 3-2. Software Part Identification, Supporting Documents

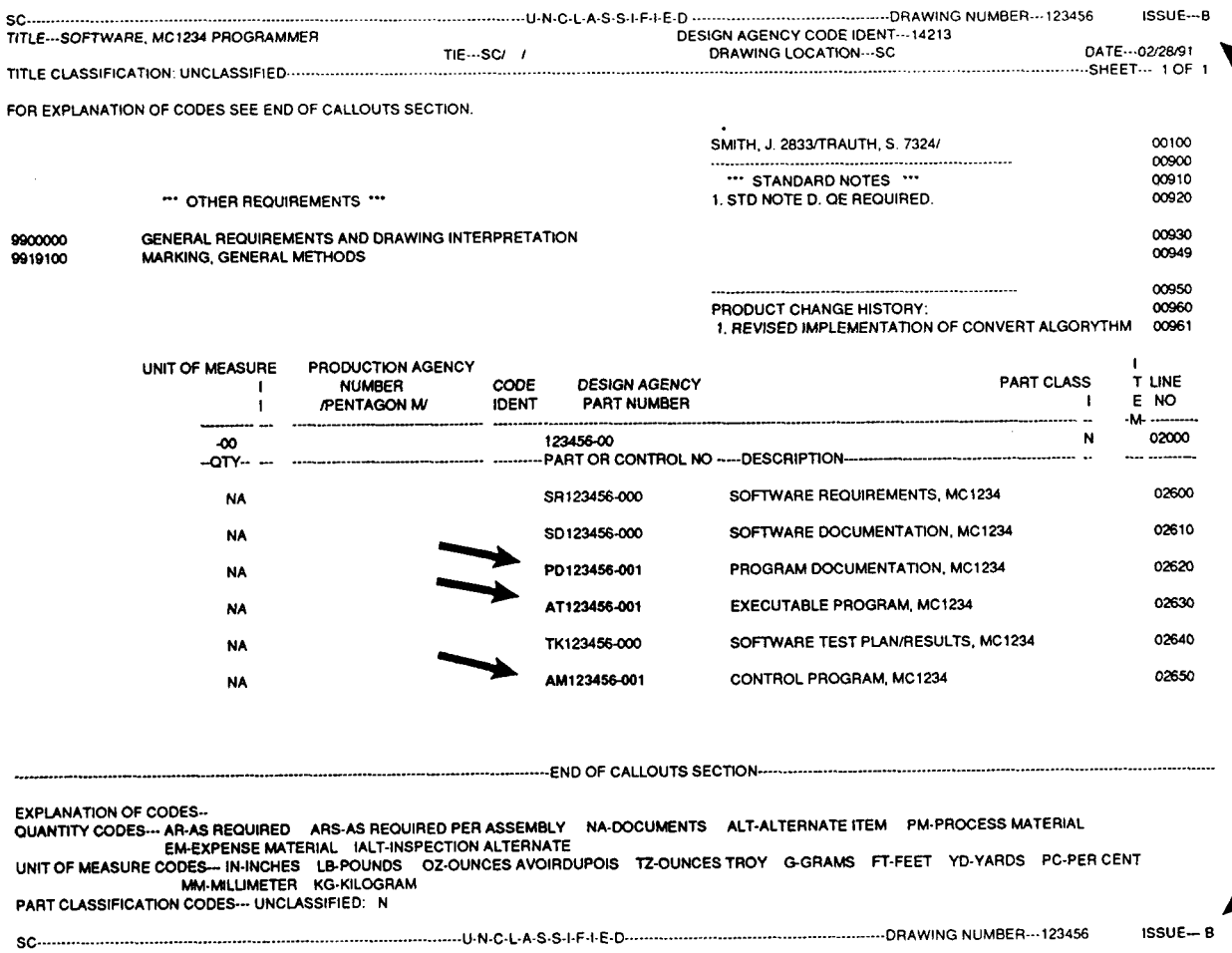


Figure 3-3. Version Change to Some Components

SC----------U-N-C-L-A-S-S-I-F-I-E-D----------DRAWING NUMBER---123456 ISSUE---C
TITLE---SOFTWARE, MC1234 PROGRAMMER DESIGN AGENCY CODE IDENT---14213
TIE---SC/ / DRAWING LOCATION---SC DATE---03/20/91
TITLE CLASSIFICATION: UNCLASSIFIED----------SHEET--- 1 OF 1

FOR EXPLANATION OF CODES SEE END OF CALLOUTS SECTION.

SMITH, J. 2833/TRAUTH, S. 7324/ 00100
---------- 00900
*** STANDARD NOTES *** 00910
*** OTHER REQUIREMENTS *** 1. STD NOTE D. QE REQUIRED. 00920

9900000 GENERAL REQUIREMENTS AND DRAWING INTERPRETATION 00930
9919100 MARKING, GENERAL METHODS 00949

---------- 00950
PRODUCT CHANGE HISTORY: 00960
1. REVISED IMPLEMENTATION OF CONVERT ALGORYTHM 00961
2. REQUIREMENTS CHANGE TO ADD SWITCH OPTION 00962

UNIT OF MEASURE		I t e m	PRODUCTION AGENCY NUMBER /PENTAGON M/	CODE IDENT	DESIGN AGENCY PART NUMBER		PART CLASS	I T E M	LINE NO
	-00				123456-00		N		02000
-01					123456-01		N		02001
--QTY--	--QTY--	---			PART OR CONTROL NO	DESCRIPTION			
NA	NA				SR123456-001	SOFTWARE REQUIREMENTS, MC1234			02600
NA	NA				SD123456-001	SOFTWARE DESIGN, MC1234			02610
NA	NA				PD123456-002	PROGRAM DOCUMENTATION, MC1234			02620
NA	NA				AT123456-002	EXECUTABLE PROGRAM, MC1234			02630
NA	NA				TK123456-001	SOFTWARE TEST PLAN/RESULTS, MC1234			02640
NA	NA				AM123456-002	CONTROL PROGRAM, MC1234			02650

----------END OF CALLOUTS SECTION----------

EXPLANATION OF CODES--
QUANTITY CODES--- AR-AS REQUIRED ARS-AS REQUIRED PER ASSEMBLY NA-DOCUMENTS ALT-ALTERNATE ITEM PM-PROCESS MATERIAL
EM-EXPENSE MATERIAL IALT-INSPECTION ALTERNATE
UNIT OF MEASURE CODES--- IN-INCHES LB-POUNDS OZ-OUNCES AVOIRDUPOIS TZ-OUNCES TROY G-GRAMS FT-FEET YD-YARDS PC-PER CENT
MM-MILLIMETER KG-KILOGRAM
PART CLASSIFICATION CODES--- UNCLASSIFIED: N

SC----------U-N-C-L-A-S-S-I-F-I-E-D----------DRAWING NUMBER---123456 ISSUE--- C

Figure 3-4. New Version, Backward Compatible

SC----------U-N-C-L-A-S-S-I-F-I-E-D----------DRAWING NUMBER---654321 ISSUE---A
TITLE---SOFTWARE, MC1234A PROGRAMMER DESIGN AGENCY CODE IDENT---14213
TIE---SC/ / DRAWING LOCATION---SC DATE---04/22/91
TITLE CLASSIFICATION: UNCLASSIFIED----------SHEET--- 1 OF 1

FOR EXPLANATION OF CODES SEE END OF CALLOUTS SECTION.

SMITH, J. 2833/TRAUTH, S. 7324/ 00100
---------- 00900
*** STANDARD NOTES *** 00910
*** OTHER REQUIREMENTS *** 1. STD NOTE D. QE REQUIRED. 00920

9900000 GENERAL REQUIREMENTS AND DRAWING INTERPRETATION 00930
9919100 MARKING, GENERAL METHODS 00949

---------- 00950
PRODUCT CHANGE HISTORY: 00960
1. NEW DESIGN BASED ON ORIGINAL REQUIREMENTS 00961

UNIT OF MEASURE / -QTY-	PRODUCTION AGENCY NUMBER /PENTAGON M/	CODE IDENT	DESIGN AGENCY PART NUMBER / PART OR CONTROL NO	PART CLASS / DESCRIPTION	ITEM NO	LINE NO
-00			654321-00	N		02000
NA			SR123456-002	SOFTWARE REQUIREMENTS, MC1234A		02600
NA			SD654321-000	SOFTWARE DOCUMENTATION, MC1234A		02610
NA			PD654321-000	PROGRAM DOCUMENTATION, MC1234A		02620
NA			AT654321-000	EXECUTABLE PROGRAM, MC1234A		02630
NA			TK654321-000	SOFTWARE TEST PLAN/RESULTS, MC1234A		02640
NA			AM654321-000	CONTROL PROGRAM, MC1234A		02650

----------END OF CALLOUTS SECTION----------

EXPLANATION OF CODES--
QUANTITY CODES--- AR-AS REQUIRED ARS-AS REQUIRED PER ASSEMBLY NA-DOCUMENTS ALT-ALTERNATE ITEM PM-PROCESS MATERIAL
EM-EXPENSE MATERIAL IALT-INSPECTION ALTERNATE
UNIT OF MEASURE CODES--- IN-INCHES LB-POUNDS OZ-OUNCES AVOIRDUPOIS TZ-OUNCES TROY G-GRAMS FT-FEET YD-YARDS PC-PER CENT
MM-MILLIMETER KG-KILOGRAM
PART CLASSIFICATION CODES--- UNCLASSIFIED: N

SC----------U-N-C-L-A-S-S-I-F-I-E-D----------DRAWING NUMBER---654321 ISSUE--- A

Figure 3-5. New Version - Not Compatible With Previous Versions

4 Making Configuration Management Work for You

In the previous chapters, several software configuration management concepts and activities were discussed, and the planning for configuration management was presented. The purpose of this chapter is to show how these concepts, activities, and plans can be applied in an effective way to your project. In particular, use of software libraries (and associated tools to generate and maintain them) and configuration control authorities, such as a Software Configuration Control Board, can help a project implement SCM. Examples are given to further clarify what level of configuration management makes sense.

4.1 Establishing a Configuration Management Concept and Plan

Configuration management need not be viewed as just another set of procedures mandated to make software development more time-consuming, costly, and cumbersome. Rather, it should be viewed as a set of procedures that will facilitate the identification and control of software baseline configurations and the configuration items within the baselines. Basic concept and planning guidelines are summarized in Figure 4-1.

Concepts and Planning Guidelines

- Plan for the proper level of SCM depending on the size and complexity of the project
- Establish an identification scheme and control structure that is applicable to the project's need (Note: The Drawing System (EP401040) and Definition of Computer Software Configuration Items (EP401045) are applied directly for WR systems and may be used in any other application. If not used, familiarity with their content is useful as a reference in establishing an independent approach)
- Identify which software products will be produced and controlled
- Identify how a software product that is under configuration control is changed during development and support
- Determine how libraries will be used to store and control software products
- Write an SCMP appropriate for the planned level of support

Figure 4-1. SCM Concepts and Planning Guidelines

4.1.1 SCM Alternative Concepts

The software configuration management plan (discussed in detail in Chapter 3) should be written as to make the software developer's and supporter's job easier, not harder. Much of the SCM concept will be dependent upon the project size - based on Source Lines of Code (SLOC), type, complexity, and criticality. Some guidelines for determining this level are presented in Figure 4-2. The SCM concepts in Figure 4-2 (Limited, Basic, Full, and Alternate) will be summarized in the following paragraphs.

	PROJECT SIZE		
Project Type	SMALL <10K SLOC*	MEDIUM 10-50K SLOC	LARGE >50K SLOC
WR/WR-Like	Basic	Full	Full
Non-WR Embedded	Basic	Basic/Full	Full
Non-WR Information	Limited	Basic	Full
Non-WR Research	Limited	Basic	Full
Non-WR Reimbursable	Alternate/Basic	Alternate/Full	Alternate/Full
* Source Lines of Code			

Figure 4-2. Guidelines for Selecting the Appropriate SCM Level

SCM Plans in the Appendices C, D, E, and F serve as examples for some of the combinations shown in Figure 4-2. The abbreviations for software products used in the following paragraphs are from EP401045: Software Requirements (SR), Software Documentation (SD), Software Source code - hardcopy (PD), Software Test Plan (TK), Software Source Code - magnetic media (AM), Software Object/Load (AT), Graphical Requirements (GR), Graphical Design (GD), and Software Quality Assurance Plan (SQ).

SCM Concept: Limited

(1) SCMP: Informal and small SCMP (3-5 pages).

(2) Software Configuration Control Board (SCCB): Informal one-person librarian; might include a higher level authority (*e.g.*, manager or project leader) responsible for higher level organization configuration management activities.

(3) Software Products: Informal SR, PD, TK, AM, AT (SD information appended to SR or embedded in PD); not released to outside customer.

(4) Use word processor for automated control tool development of any software written documentation. Drawing tool used for automated development and control of graphic documents such as GR and GD supplements to SR and SD. Hardcopy and electronic media form of documents maintained by Librarian.

(5) Use source code automated control tool during development. Software project leader assumes Librarian role. Software released directly to user from project library (separate from the personal library). Final form of the project library is the initial release library. Contents are not typically submitted to the Sandia Drawing System for archival/backup purposes. Separate copy maintained by the Librarian for reference (and possible use during follow-on support activities).

(6) Informal problem reporting and status accounting during development and support.

(7) Archival for backup/recovery is on an informal basis controlled by the individual/team development standards.

SCM Concept: Basic

(1) SCMP: part of SQAP, or small SCMP (3-5 pages).

(2) SCCB: Small 2- to 3-person software review authority

(3) Software Products: SR, PD, TK, AM, AT (SD information appended to SR or embedded in PD).

(4) Use word processor for automated control tool development of SR, TK, SQAP/SCMP documents. Drawing tool used for automated development and control of graphic documents such as GR and GD supplements to SR and SD. Hardcopy and electronic media form of documents maintained by Librarian (Software Project Leader or designee) after inspection.

(5) Use source code automated control tool during development. Software project leader assumes Librarian role. Software released directly to Librarian (project library) from the Developer (personal library) after inspection and unit test. Final form of the project library is the initial release library and contents are submitted to the Sandia Drawing System for archival/backup purposes. Separate copy maintained by the Librarian for reference (and possible use during follow-on support activities).

(6) Informal problem reporting and status accounting during development and support.

(7) Sandia Drawing System used for archival and change control after delivery.

SCM Concept: Full

(1) SCMP: separate full SCMP.

(2) SCCB: minimum of system project leader, software project leader (chair), Librarian, test engineer, and the software technical leaders as needed. Involve division managers of organizations. SCCB part of a larger System CCB. The control authority would be in accordance with plans and standards that might bridge more than one organization. SQAP would reference the SCMP.

(3) Software Products: SQ, SR, SD, PD, TK, AM, AT

(4) Use network control of project resources (word processor, source code control, problem reporting and tracking, project management, other support tools: compilers, linkers). Word processor used for automated control tool development of SR, TK, SQAP/SCMP documents. Drawing tool used for automated development and control of graphic documents such as GR and GD supplements to SR and SD. Hardcopy and electronic media form of documents maintained by Librarian (Software Team Member) after inspection.

(5) Use source code automated control tool during development. Software Librarian role may be full time or a shared role by one of the software developers. Software products formally released to Librarian (project library) from the Developer (personal library) after inspection and unit test. Final form of the project library is used for system/integration test: Changes are controlled through a problem report/change form. Project library and contents are submitted to the Sandia Drawing System for archival/backup purposes. Separate copy maintained by the Librarian for reference (and possible use during follow-on support activities).

(6) Rigorous problem reporting and status accounting during development. Any changes to software products released to the project would be processed through the SCCB using a change control form.

(7) Sandia Drawing System used for archival and change control after delivery.

SCM Concept: Alternate

(1) SCMP: dictated by customer (typically DOD-STD-2167A SCMP).

(2) SCCB: dictated by customer; customer may have a representative on the SCCB depending upon the project size. Similar to Basic or Full concepts. Control authority would be in accordance with organizational plans and standards that might bridge more than one organization. SQAP would reference the SCMP.

(3) Software Products: Identification might vary, but would still be similar to SNL system: SQ, SR, SD, PD, TK, AM, AT.

(4) Use network control of project resources (word processor, source code control, problem reporting and tracking, project management, other support tools: compilers, linkers). Word processor used for automated control tool development of SR, TK, SQAP/SCMP documents. Drawing tool used for automated development and control of graphic documents such as GR and GD supplements to SR and SD. Hardcopy and electronic media form of documents maintained by Librarian (Software Team Member) after inspection.

(5) Use source code automated control tool during development. Software Librarian role may be full time or a shared role by one of the software developers. Software products formally released to Librarian (project library) from the Developer (personal library) after inspection and unit test. Final form of the project library is used for system/integration test: Changes are controlled through a problem report/change form. Project library and contents are submitted to the Sandia Drawing System for archival/backup purposes. Separate copy maintained by the Librarian for reference (and possible use during follow-on support activities).

(6) Formal problem reporting and status accounting during development, with the customer interaction. Problems would be reported using a problem report form.

(7) Sandia Drawing System used for archival and change control after delivery.

4.1.2 Plan to Use the Sandia Drawing System

The Drawing System identification scheme (EP401045) discussed in Section 3.4 is typically associated with WR and WR-like software. It should be used for all software called out in DOE Order 1330.1C unless a specific project requires something else. This means all controlled software product items will have an identification number consisting of a two-character, software-type prefix, a six-digit part number, a three-digit revision (correlation) suffix, and an issue letter. For example, SR123456-002, Issue J would represent Version 2, Issue J of the requirements specification document for software part 123456.

4.1.3 Identify Software Parts and Control Authority

A major part of the SCM concept and planning stage is to identify which software component parts will be controlled: documents, source, object, data base, data files, support equipment, and so forth. Minimally, it is recommended that the software include a requirements specification, design document, test plan, source and support data files, and an executable copy of the software code.

Those persons with the authority for establishing software product baselines and formal approval of changes to that baseline should be identified. During software development, it will probably be the system project leader or the software project leader. It may also involve a customer representative and/or a supplier representative. During support, it will depend upon who has responsibility for the software support function and how the Sandia organizational configuration management concept is implemented (see Appendix C). The person with the final authority on software changes should be the SCCB (or software configuration review board) chair. A software Librarian should be appointed from among the software team members to control developmental releases of the software product and provide information for the SCCB. Other SCCB roles should be identified as necessary.

4.1.4 Identify SCM Relationships

Identification of the relationships among the customer, user, developer, supporter, and any supplier is critical to the success of SCM. If the customer is internal to Sandia, such as an Information Resource Management group, or a Systems Engineering group, then representatives of each interest group should be assigned to the SCCB. If the customer is external to Sandia, then ensure the customer has adequate review and approval of the SCMP. Acquaint these customers with the software change process so that they can submit change requests. These change requests may be a result of major program reviews, demonstrations, failures during system operation, or may be desired enhancements. Any software suppliers for a Sandia system should provide evidence that the supplier's own internal software configuration management satisfies Sandia's requirements. Supplier requirements are stated in the SCMP. Minimally, procedures for delivery and installation of standard vendor software updates, and expected vendor responses due to any vendor software failures should be a well-defined part of the SCMP.

4.2 Establishing Change Control Procedures and Authority

Once the configuration management concept has been established, the next step toward implementing configuration management in a sensible, painless way is to define change control

procedures and the associated change authority level. The interrelationships among change control procedures, change authority, and baseline configurations are described in this subsection. Baseline configuration items are stored in libraries: Personal, Project, and Release. Details of the use of libraries are described in subsection 4.3. A basic list of guidelines is illustrated in Figure 4-3.

Change Control and Authority Guidelines

- Define required change control procedures and baselines - Allow for flexible change control paths for time (normal, emergency) and change type (major, minor)
- Define personnel positions responsible for SCM, including the software Librarian and the SCCB - Identify control level
- Define SCCB group logistics and meeting procedures
- Run SCCB meetings in standard fashion: identify issues, assign responsibility, make accept/reject/defer decision
- Incorporate libraries and associated automated tools to assist the change control process and provide a reporting link to status accounting

Figure 4-3. SCM Change Control and Authority Guidelines

4.2.1 Change Control Procedures

Change control procedures are dependent upon establishing a baseline configuration to which changes can be applied. The recommended baselines as they evolve during the software life cycle include:

(1) **Functional:** configuration items include the system functional documents that describe the system operational concept, the system functional requirements, and any manufacturing environment system requirements. It is from these system documents that software requirements are specified and/or derived.

(2) **Allocated:** configuration items include the software requirements specification, the hardware requirements specification, and necessary external and internal interface requirements specifications.

(3) **Developmental (Design):** configuration items include the software requirements specification, the design specification, and the test plan. Other documents could include the software quality assurance plan and the software configuration management plan.

(4) **Developmental (Implementation):** configuration items include the Developmental (Design) baseline plus the software source/object code and the unit/component test cases/results.

Other documents could include the maintenance plan, user guide, operator manual, and so forth.

(5) **Product:** configuration items include all software products that will be released to the customer, typically the Developmental (Implementation) baseline plus the system test cases/results.

(6) **Product (Version/Issue Updates):** configuration items include the Product baseline plus the changes to individual product baseline items.

The change procedure involves coordination of change requests across the various levels of change authority to ensure that the baseline configuration items can effectively evolve to include corrections, enhancements, and adaptations to changes in the operating environment. Elements of the change procedure are depicted in Figure 4-4.

4.2.2 Change Control Authority

Change control authority depends upon the level of desired control. For software items being controlled at the lowest level by individual persons as part of their personal library, the responsible individual is the only authority. For promotion of items from the personal library to the project library, the items should pass a software inspection and (if it exists) the approval of the Software Configuration Control Board. After promotion to the project library, any changes must be approved by the software inspection team and/or the SCCB. Promotion of the software items in the project library to the release library (approved for delivery to the customer) is the responsibility of the project manager and the project CCB (if it exists). The CCB and the SCCB may be the same on small projects.

Levels of Change Authority

A key concept to understanding effective use of SCCBs is that of different levels of authority or control. The level of authority may refer to the personnel or the procedures required for approval. Levels of control should be established for various kinds of changes to each of the different classes of libraries. The types of changes that require different levels of approval authority should be clearly identified. Procedures involving the approval of changes by an SCCB should then be established for each promotion path between libraries and for each type of change, forming a progressive hierarchy of approval levels as a project builds a software product. These procedures might require that more than one SCCB be established for various types of changes at a given level or that a single SCCB have responsibility for more than one approval level. For instance, inclusion of a minor error correction into a library might require approval from a different SCCB than the one that would grant approval to the addition of a major new feature. The approval procedures required for minor, major, emergency, or normal changes should vary within the same SCCB structure. The key is to provide required response efficiently and effectively so that changes are implemented correctly in a timely manner.

Levels of authority may vary with the software's development life cycle phase. For example, changes to software in an early developmental stage typically require a lower level of authority for

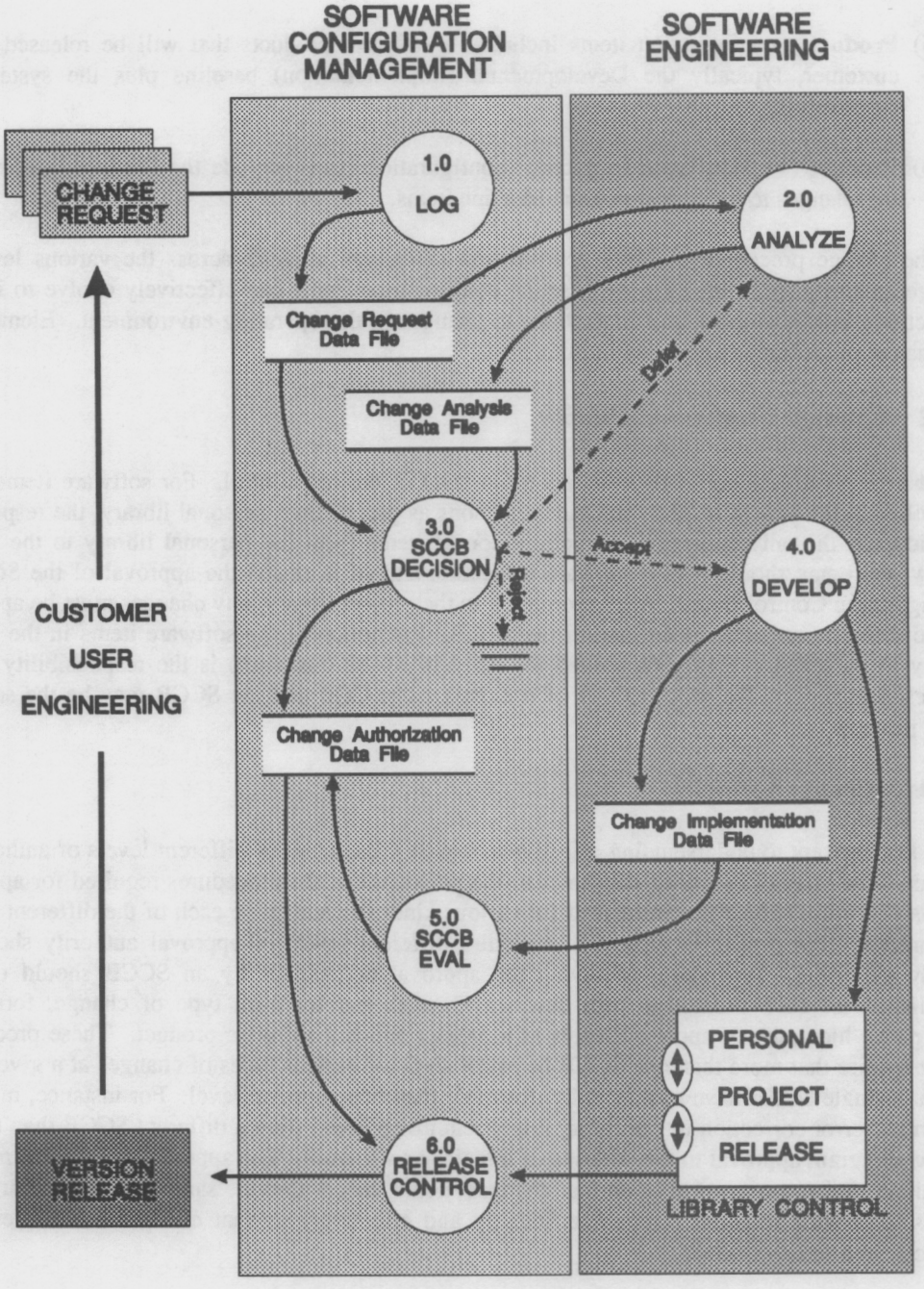


Figure 4-4. Change Control Procedure

approval than changes made after release, when the potential impact is much greater. The larger the impact of a change on developmental resources, schedule, number of users, software functionality, or interfaces, the greater level of authority that would normally be required.

Characteristics of SCCB Members

The SCCB is dependent on the size and nature of the development project, and on the level of approval required for controlling a change to a specific library. For small projects, a single individual authority level may provide satisfactory control. This individual may be the developer (or cognizant manager, if some level of management control is desirable), and might be given the authority for approving and promoting changes for any of the libraries. At the other extreme, a very large and critical software development project might require two or three (or perhaps even more) different SCCBs, each composed of several people drawn from appropriate technical staff and/or management, with different levels of authority and control. The key point to be made here is that the membership of an SCCB must sensibly reflect the needs of the software project for control of changes to a library at a given level.

SCCB members must be knowledgeable on the important factors to consider in approving a change such as technical, economic, or political issues. For changes that involve a significant technical component (e.g., modification of a numerical solution algorithm), approval should be made by an SCCB with substantial technical expertise and with sufficiently intimate knowledge of the details of the modification to pass judgment on it. This SCCB could be the developer or group of developers on the project if adequate assurances can be made that the review process leading to approval is sufficiently independent. Often the development team is the only source of such technical expertise and knowledge of the software.

For changes that involve questions normally reserved for management to answer, such as whether a major new feature should be added based on schedule/cost risk or political considerations, approval should be made by an SCCB with corresponding management representation and authority. Obviously, the larger the impact of the new feature (economically on the project, politically on the user community, or technically on the software functionality), the higher the SCCB authority level that may be needed.

SCCB members should have a vital interest in other organizations or customers that interface with the software products. For instance, change to a software product that may affect the function of another piece of software or hardware should be reviewed by someone responsible for that related product. If a new feature not specified by the requirements is desired to be implemented by the developer, the customer should be involved in approval of that change. Such interfaces must be considered prior to change approval. Appropriate representation on an SCCB by personnel from interfacing organizations and customers provides a formal means to ensure that effort is not wasted because of unforeseen side effects or miscommunication.

Interfaces among various SCCBs may exist, both within the project and with other organizations, and these should be clearly defined. In particular, identifying lines of communication, areas of responsibility, and relative levels of authority are each issues that should be explicitly spelled out.

SCCB Functions and Change Procedures

The exact function or role carried out by a SCCB and the process by which it accomplishes its function must also be carefully considered. A software project should have established one or more mechanisms to generate and/or receive Software Change Requests (SCRs). The project should identify how such requests receive preliminary approval for resolution, and what role the various SCCBs should play at this point. For a major software change, it may be necessary for a management-level CCB to grant preliminary approval before resources are committed to resolving the change request. For minor changes that do not require substantial resources, it may be entirely adequate for a change to be made in a developer's personal library before consideration by an SCCB for promotion to a project library.

Interim approvals by an SCCB may also be necessary for lengthy, time-consuming changes. Following preliminary approval to proceed with development of a plan to resolve an SCR, considerable effort may be required just to develop that plan. Interim approval of that plan by an SCCB may be desirable before proceeding with plan implementation (i.e., writing or modifying the code). Revision of the plan may require additional interim approvals.

An SCCB must grant final approval for implementation of a particular software change into a baseline library. The nature of the review required for approval should be clearly defined. For some informal projects, it may be sufficient to perform a broad top-level review without examining the technical details down to the line-of-code level. For critical projects, it may be necessary to conduct a detailed line-by-line software inspection to give reasonable assurance that the new coding is fault-free and does not adversely affect the performance of other parts of the software. The nature of this review process, which also may vary according to the type of change being made, also suggests additional criteria may be required for appropriate level of authority and makeup of the governing SCCB.

Procedures for approving third-party or off-the-shelf software, for use within a project, need to be specifically addressed by an SCCB. Such software may include public domain, vendor supplied, vendor supplied but modified, subcontractor developed, proprietary, or reusable software developed by another project. The extent to which such software must be checked, tested, documented, and updated by later third-party changes should be defined.

Other SCCB procedural issues to be resolved include disagreements within the SCCB, frequency and duration of review meetings, timely review of SCRs and changes, and so forth. The answers to these questions all depend on the other considerations raised in this section; there is no single answer, other than to do what seems to make the most sense given the particular needs of the software project.

4.3 Using Baselines and Libraries

Once the configuration management concept, change control procedures, and change control authority levels have been established, the software project must make effective use of libraries. Guidelines for the use of libraries are summarized in Figure 4-5.

Use of SCM Libraries Guidelines

- Identify the levels of libraries to be used. Recommend use of personal, project, and release libraries as a minimum for all projects.
- Define libraries for documentation (for word processor forms of software products), and source code management (for source, data, build form of software products).
- Define the promotion procedures between library levels, including interface with the change authority and the change procedures.
- Incorporate libraries and associated automated tools to assist the change control process and provide a reporting link to status accounting.

Figure 4-5. SCM Libraries Guidelines

Libraries are simply collections of files (usually electronic) that contain the various software component products. Libraries provide the means for identifying and labeling software components and for tracking the status of changes to them. The trend is for increasingly sophisticated Computer-Aided Software Engineering (CASE) tools to be used to automate the management and use of libraries. The extent to which these tools are advantageous or necessary depends on the level of complexity of the software product and the programming environment. Examples of libraries include:

(1) **Word Processor System:** to control text parts of software documents

(2) **Graphics Processor System:** to control graphic drawings associated with the software documents

(3) **Source Code Management System:** to control source code, source code internal relationships, data, object builds

(4) **Change Request Tracking System:** to control the logging of change requests and tracking of their change status (open, closed, deferred, rejected) and implementation in releases.

One principal function of software libraries is the implementation of a configuration control scheme for development baselines. These baselines evolve throughout the development process until the final developmental baseline becomes the product baseline released to the customer. This same scheme can be used during the support phase to process changes to the product baseline. This identification scheme depends upon the underlying structure of the software as well as the programming environment and SCM tools. The libraries thus define the status of the software configuration at any time by referencing a beginning point (system functional baseline version), establishing the initial software baseline (allocated requirements baseline), and the various stages of the development process through development baselines that contain any modifications that have been made since the baseline was defined. They also provide a means for relating file nomenclature

(*e.g.*, file names for software units and components) back to the configuration identification scheme (*e.g.*, the software unit, component, and configuration items names).

Software libraries are used as tools to manage the configuration evolution of the software development baselines such that changes are made in a systematic, controlled way and that the configuration is not inadvertently (or purposefully) corrupted. This ensures that the status of the software is well-understood at all times and that changes have not been made without having gone through a process of review, approval, and authorization for implementation, producing an audit trail. Libraries can also serve as means to control access to software components, with standard procedures to be followed and checks made via passwords or authorization lists to be able to successfully modify them. Thus, a certain amount of safety and protection (against loss or deletion, accidental or intentional) is afforded in a convenient way to software components through effective use of a library structure.

Levels of Libraries

The number and types of libraries will vary from project to project according to the size and complexity of the software product and the organization and specific needs of the software developers. There are fundamentally three levels of libraries as described in Section 2.2; personal library, project library, and release library. They serve different functions during the course of software development.

The degree of formality governing the access to libraries and the authorization for changes to them depends heavily on the nature of the project and on the software itself. For a small project, often involving only a single code developer, less formal procedures can be established to access and use the various libraries. However, many developers find it difficult to exert the self-discipline necessary to keep their work orderly, with little thought given to baselines and systematic change. Therefore these procedures, however informal, should be clearly stated in an SCM Plan and then followed.

Example: A Small Research Project

Consider a developer working on a very small analysis program of perhaps a thousand lines of code. Once the program is capable of functioning in some minimally acceptable way (e.g., successfully executing a set of test problems) the code may be promoted to a controlled project library status, forming the first internal baseline version of the program. Subsequent modifications in response to further testing and debugging or addition of new capabilities (made to copies of the project library in the developer's own personal working space) are carefully delineated, justified, and systematically incorporated into the official project library. These modifications form new internal baselines when criteria established for promotion are satisfied (*e.g.*, successfully repeating previous tests or executing new ones).

It is very important that the personal library working space be kept separate from the project library, for example in different directories. When a version of the program satisfies all requirements established for it, the project library may once again be promoted to official release library status. Once this happens, the new version of the code must not be changed and the release baseline should be archived in some fashion. Additional modifications necessary after release will

form the basis of another version. This kind of development is typical of research and development environments.

Example: A Large Software and/or WR Project

Many projects are much more formal by virtue of their size or the critical nature of their software. For very large software development projects, work must be carefully coordinated to ensure that coding modifications do not alter carefully defined interfaces or otherwise affect the function of other software units. If multiple developers work on the same portions of the software, additional SCM controls must be established to prevent the possibility of incompatible or lost changes. For example, if two developers modify the same routine in parallel for different reasons, some means must be found to identify and reconcile any differences before the modifications are incorporated into the official project library.

All software that resides in weapons or is part of weapons-related systems would be considered critical software. Probably the most important aspect of critical software development from a configuration management standpoint is that there should never be just one person involved. At the very least there should be a developer and a reviewer. Typically there are several people on a team assigned to develop and test critical software. In critical software development programs, corrupted software can be the cause of equipment failure or personal injury or death. In this case, password control at one or more levels is usually the minimum protection provided the library.

Many operating systems allow the creator of a file to specify which users can access files and at what level (e.g., read only, read/write, no access). Furthermore, some systems are capable of tracking file access. Limiting access to libraries through these means can be a reasonably effective method to control who can have a "need to know" access, and prevent inadvertent changes to the software products. Most classified machines have more strict administrative procedures and automated protection mechanisms for access control. Few have any kind of external link.

4.4 Managing the Release and Concerns After Release

There are some special concerns that should be addressed during the software development process to facilitate the management of the initial software release and subsequent software releases during the software support phase. Some of these concerns include:

(1) Transition of SCM to support activity

(2) Changes to application system software

(3) Changes to vendor equipment and software

(4) Reuse of software in other applications

Transition of SCM to Support Activity

The support activity is simply the organization that will have the responsibility for supporting the software after its initial product baseline release delivery to the customer. This support activity

may be the original software development group, or some other group defined by the customer. The transition of the SCM function to the support activity should be addressed in the SCMP. It is critical that this transition be clearly understood early in the software development life cycle. It is very difficult to transition a software system to a support activity that has different computer equipment, library systems, identification schemes, change control procedures, and personnel who are not familiar with the software. Since some of this may occur (see the next few paragraphs) even if the development group is the intended support activity, plans for possible evolution should be integrated into the transition concept.

Establishing a compatible software support environment is a critical part of the transition of SCM to the support activity. The initial support environment should include computer equipment, system support software, test equipment, and simulation/emulation capabilities that are as close as possible to the development environment. There may be security concerns that must be addressed to provide separate facilities for classified software, data, or testing.

Changes to the Application System Software

Changes to the application system software are the primary concern of the support activity. Planning for those changes and integrating the change concepts into the SCM transition are important parts of the SCM development activities.

The SCM activities are intertwined with and a major part of the software support concept. One of the activities during development (and also during support) that facilitates planning for SCM is establishing an estimated change profile. In order to estimate resources required for the SCCB, Librarian, and other configuration management activities, it is necessary to understand how much change activity is expected. This change activity should include an estimate of when future software releases can be expected and what level of changes will be included. Changes can be corrections, enhancements, or adaptations to environment changes, so the expected change activity should be stratified across these change types. Typically, most organizations can use historical data or heuristic estimations to complete the projected software change profile. Agreement among the customer, support activity, and development team is also an important part of establishing the projected software change profile. This type of "Software Support Concept" information should be included in a project computer resources integrated support plan or perhaps an integrated logistic support plan.

Changes to Vendor Equipment and Software

Changes to vendor equipment and supporting software can have a major impact on application software systems. It is critical to control configuration of this equipment if it is critical to either the continued support or operation of the application software. Products that should be controlled include:

(1) Computer system models used for support

(2) Operating systems

(3) Compilers, linkers, run-time support software used to build the production software load

(4) Computer system models used for operation

(5) Test equipment and software used in the unit, component, integration, and/or system test of the software

(6) Any software developed on subcontract

(7) Written documentation on any of the above products

A related issue is the availability of such vendor equipment and software. If the vendor goes out of business or is absorbed by some other business, what is the availability of the critical vendor components upon which the application software systems is dependent? For example, if a real-time operating system from vendor A is embedded in an application software system, and vendor A goes bankrupt, how are future copies of the operating system obtained? How are changes to the operating system accomplished if a fault is discovered? Will the source code be available (perhaps for a price) so the support activity can assume its support? These issues should be addressed in the Supplier Control subsection of the SCMP. This subsection includes discussion of any Subcontractor Software and any Vendor Software. Further issues to consider are described in IEEE Std 1042-1987 [IEE87].

Reuse of Software in Other Applications

Perhaps the only way in which software productivity will be significantly improved (one or two orders of magnitude) is through reuse. The SCM activities of identification, change control, status accounting, and audit are all major contributors to improving the chances that application software can be reused. SCM provides evidence that a software product is well-defined and controlled. This information allows for efficient analysis of the software for potential reuse applications.

The ability to retrieve software products after release, adapt the software products to another application, and redefine the new application baseline is a direct function of software configuration management activities. In some cases, it may even be necessary to reuse a previous version of an application system software product. In this case, it would be necessary to be able to retract version changes - probably using automated tool library management capabilities. If the automated tools, versions, and changes have all been controlled through software configuration management activities, then the chances of recovering a previous software version are good.

4.5 Managing Configuration Management Records

There are many records that must be managed as a part of the software configuration management process. Some of these records include:

(1) Change Requests

(2) Change Request Analysis

(3) Change Authority Decisions on Change Requests

(4) Minutes of SCCB Meetings

(5) Status Reports Showing Change Request Status

(6) Version Control Documents

(7) Baselines of the Software Products

(8) Baselines of the Software Support Environment

The management of software configuration management records is important for at least two reasons. First, the success of software configuration management is dependent upon these records. Without careful recording and control of these records the status of any software product's progress from library to baseline or baseline to baseline is not known. Schedule deadlines for releases can not be met, and released products have little chance of satisfying customer requirements. Second, the success of future software updates depends on traceability to current and previous baseline versions. These records provide that traceability.

Several issues are important to consider in the management of such records. What will be the record media? Will the records be on-line or off-line? How long will the records be maintained? In the case where records are updated (*e.g.*, versions of software), how many versions will be maintained? Where will the record media be maintained? For what use are the records being maintained (engineering, software quality, customer/legal, backup/recovery)? Perhaps the key questions are: Can the software be recreated from the maintained records? Can the history of the software release be retrieved if necessary? Can the records be used to improve the software life cycle processes?

There are automated tools that provide assistance in managing all project change requests (including software-specific changes). These tools allow for input of the information from a change request form or analysis form, reports on status information as input to the SCCB, summary status information across all change requests or stratified by user-specified queries, and tickler files for critical dates. It is recommended that such tools be used in conjunction with other library automated tools to provide assistance in the management of software configuration management records. The following chapter provides guidance on evaluating and using these tools. Since such tools change frequently, it is also recommended that project personnel contact the Software Quality Engineering organization for latest guidance on availability of these tools.

5 Considering Tools

The key point is that automation tools should support SCM and make it easier. They cannot guarantee its success nor does the lack of such tools ensure its failure. A specific automation tool should not be used if it becomes too burdensome or interferes with the development process for a project. Therefore, different types of projects may need different types of SCM tools. Refer to [SSGv5] for a discussion of tools used at Sandia.

5.1 Goals for Automation of SCM Functions

The basic goals of SCM are higher quality software development, faster and cheaper software development, and easier and cheaper maintenance. SCM helps meet these goals in two ways. First, it keeps track of which software components are used where, when and how they are changed and the effects of the changes, and which versions are in which stage of release. These actions support and ease the administrative detail for large software projects. These actions also minimize the rework and corrections required by uncoordinated changes. Second, it provides the support necessary for reusable software. Unless software items are clearly identified and changes to them are controlled, reusability is difficult if not impossible.

SCM consists of four distinct functions: configuration item identification, change control, status accounting, and audits and reviews. Automating SCM affects these functions differently. However, the overall reasons for automating parts of SCM include: improved use/access to common data, improved accuracy of common data, and to relieve people from tedious, boring, and repetitive operations so they will be available for the more labor-intensive tasks of SCM. For example, keeping track of software entities and changes can easily be automated and allows people time for tasks such as audit and review.

Configuration item identification involves selecting or defining a classification scheme, coding the software items using the scheme, and loading this information into the SCM system. Developing the classification scheme requires the most effort within this function. Automation can only help this activity if an SCM tool is selected that has a built-in classification scheme. This would only occur with a tool specifically designed for software configuration management. If a generic configuration management tool, such as Sherpa Design Management System (DMS), were selected, then a specific classification scheme for SCM would have to be defined and loaded into the system. Assigning identification numbers and loading the data into an SCM system could be tedious if it were done retroactively. In most cases, however, the classification scheme is defined at the start of the project and identification numbers are assigned and loaded incrementally as new items are created during the development process. The types of items that must be included in the classification scheme are those identified in Section 5.3 as software entities.

The second SCM function is change control. This function includes: generating a change request, evaluating the impact of the change request (perhaps in conjunction with other related change requests), approving the request, making the approved changes, and tracking the new versions of the changed items. Most of this function requires human action. Automation cannot

help evaluate or approve a change request. The only way automation can help this function is to keep track of and provide the information needed to support administration of the change control process.

The status accounting function is the one where automation can dramatically help. Reporting and analysis become much easier if there is an automated tool with a data base to support software configuration management.

The audit and review function is the most labor-intensive part of SCM, in some cases requiring more resources than the other three functions combined. Unfortunately, automated tools can provide little support for this function beyond tracking its administrative data.

5.2 Framework for Evaluating SCM Tools

Historically, software configuration management tools have usually managed only specifically defined types of baseline documents or have only managed code modules.

As long as these baseline types of documents were only visible to the developer working on them, they were not under the SCM tool environment. At some point there was a request to release them to other project developers and/or to users. At this point they came under software configuration management. SCM was only interested in documents from this point and beyond in the development process, and then only with a specific, predefined subset of the document types, not with all types of documents. Much of the work involved manual manipulation of files and perhaps some in-house software to aid the process.

This approach was developed before CASE tools became widely available. CASE availability permits a more comprehensive approach to SCM. A CASE tool should support all types of working documents for the software development and support process. Some of these documents are created and/or used by the SCM CASE tool itself, other CASE tools, or other applications such as word processing for various documents and manuals.

Without a CASE tool, only the traditional approach to SCM is possible. There are two types of traditional software configuration management tools — SCM-specific and generic. The key difference is that an SCM-specific tool, which was developed explicitly for software configuration management, should already have a predefined standard set of software entities which it manages. It should also have a predefined standard set of promotion/release levels. These predefined concepts should provide a basic level of identification and control to allow an organization or project to get started with SCM. However, since most organizations have their own variations, even an SCM-specific tool should allow tailoring. It should be possible to add to, subtract from, or rename the predefined concepts/documents. At a minimum, the SCM tool should support the software entities defined in Section 5.3

Before selecting an SCM tool, the organization should have analyzed its requirements, determined how it wants to do SCM, identified the software entities that must be managed, and defined the promotion/release levels. Then the organization can evaluate the specific SCM tools to see if they can support the specific SCM environment or can be tailored to support it. If available

SCM tools cannot support the environment, then much of the analysis needed to take the alternate approach, customizing a generic CM tool, has already been done.

Most of the generic CM tools were developed to do configuration management in a manufacturing environment. Therefore, they are designed to support a well-understood, long-standing approach to configuration management. They are also generic and easily tailorable because they must support a wide range of manufacturing industries. For example, Sherpa DMS allows tailoring to define deliverable/document types, promotion levels, and projects. It can also associate specific types of documents and promotion levels to specific projects.

In general both an SCM-specific tool and a generic CM tool should provide the same capabilities. The main difference is in the amount of tailoring required. These capabilities include:

- Defining different types of documents
- Defining different projects
- Defining different promotion levels for different type documents and/or different projects
- Relating types of documents to specific projects or types of projects (especially important because different types of projects - by size and/or discipline - may have different SCM rules
- Defining the Software Configuration Control Board (SCCB) to promote/release documents
- Assigning different SCCBs to different projects and/or document types

For smaller projects and those without CASE tools this approach to SCM, managing baseline documents manually or with either a software-specific or generic CM tool, is appropriate. Even with larger projects or those using several stand-alone CASE tools, this approach may be used initially as part of the learning process.

However, in an integrated CASE environment, SCM should be an integral part of the CASE tools. A data base should support the CASE tools and allow them to share common design data. This data base should also include release status and version information which the various CASE tools, including SCM, should be able to access and use.

The following is a summary tool taxonomy for CASE tools supporting SCM:

Level of functionality and integration

(1) Tools that only support/manage (*e.g.*, control access to and promotion of) documentation. (This is especially true of general configuration management tools adapted to SCM. Traditional SCM only involves this level of functionality.)

(2) Stand alone SCM tools designed to support the SCM functions, but not the other CASE type of functions.

(3) More general CASE tools which also support the SCM functions. (The additional CASE functions supported may be either front-end design tools or back-end code generation tools.)

Classes of tools by SCM function

(1) Configuration identification and control:

Version tracking and change control
Library data base
Library scripts

(2) Configuration Status Accounting (CSA):

Standard CSA reports and forms
CSA data base
Report generators

(3) Configuration Audit:

Traceability Matrix

5.3 Software Entities that Tools Must Control

A Software Configuration Management tool should manage all of the software deliverables and the entities or objects needed to completely define these deliverables. This section identifies the types of software deliverables, the software entities and objects, and the relationships among them. It does this using an information model, created using a Nijssen Information Analysis Model.

Figure 5-1 shows the basic taxonomy of objects that must be managed by configuration management. A system is composed of one or more subsystems such as mechanical, electrical, and computer or information systems. These computer or information subsystems have both a hardware and a software component. The software subsystem has many components which are related to each other, to their documentation, and to the hardware components on which they run. These components are identified, related to each other, and tracked by SCM. Software components can be either application software or system support software. Application software is software that satisfies a set of end user requirements. Support software includes operating systems, communications, Data Base Management Systems (DBMSs), and/or utilities.

Both types of software can be further decomposed in terms of both functions (or processes) and data. On the function side, there are programs, which are composed of modules which may be further decomposed into additional modules. There are also interfaces between modules. On the data side, there are information models, data models, and data base schemas (or in non-data base systems, "file formats").

Both these function and data objects are real things directly involved in the system development process. For example, programs, modules, and data base schemas are stored in the computer in

source and object form. There are also documents that further describe or explain these objects. Tools must manage the various types of objects, their documentation, and how the different versions of each are related to the others.

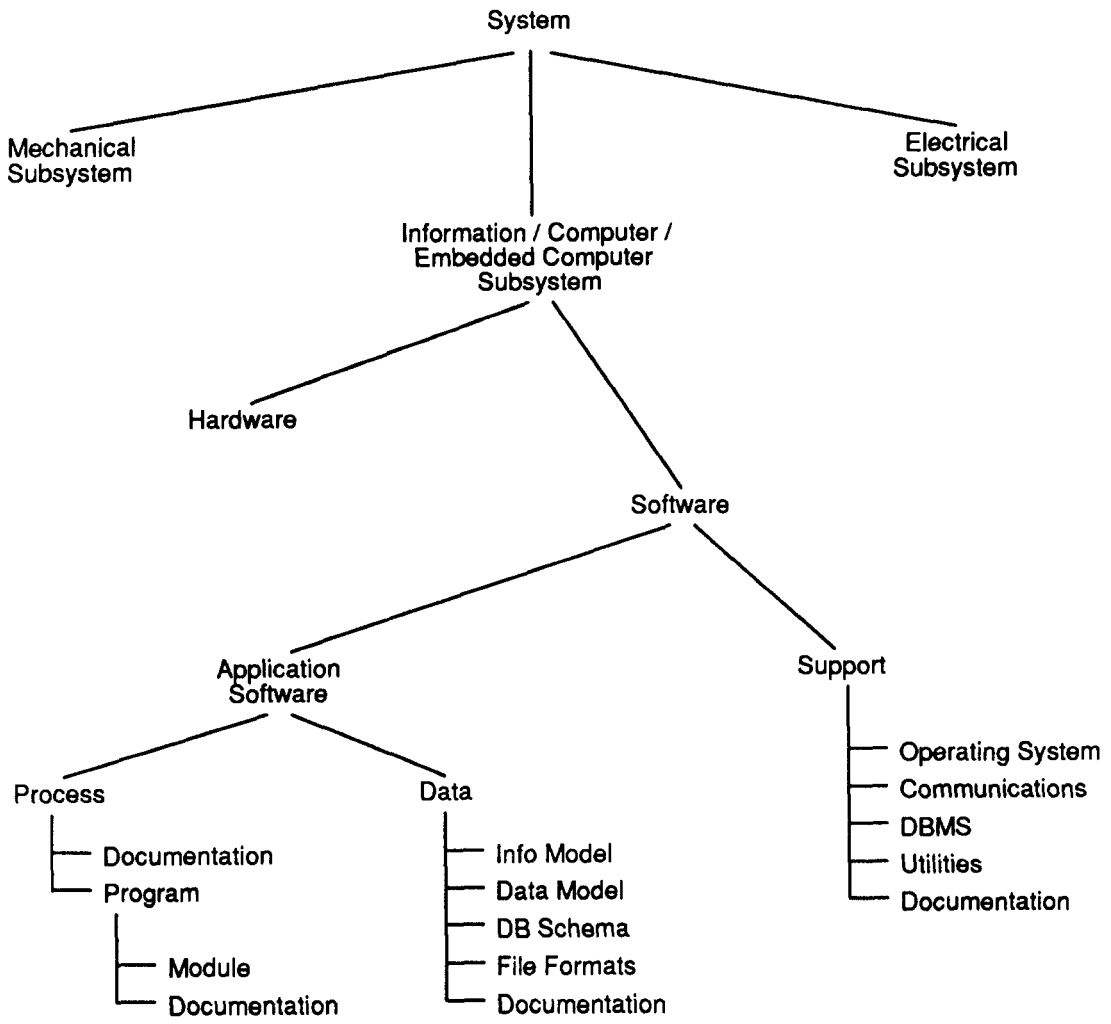


Figure 5-1. Taxonomy of Objects

Figure 5-2 shows the information model that relates the various types of software entities or objects. At the highest level there is the software library. There are the three distinct types of libraries: dynamic, controlled, and static (or personal, project, and released) which differ primarily in how changes are controlled as discussed in previous chapters. Each library has an identifier and a person responsible for it. At the lowest level, a library consists of units which are included in a library. Each unit is identified by a unit id and a version number. A unit also has a release status, an effective date for the status, a person who created it, and a person who is responsible for it

(which may be its creator or someone else). A unit has an operating environment, which has both hardware and software components. There are several subtypes of units. A unit can be either code, data or documentation. The code or data element can be in either source or object form.

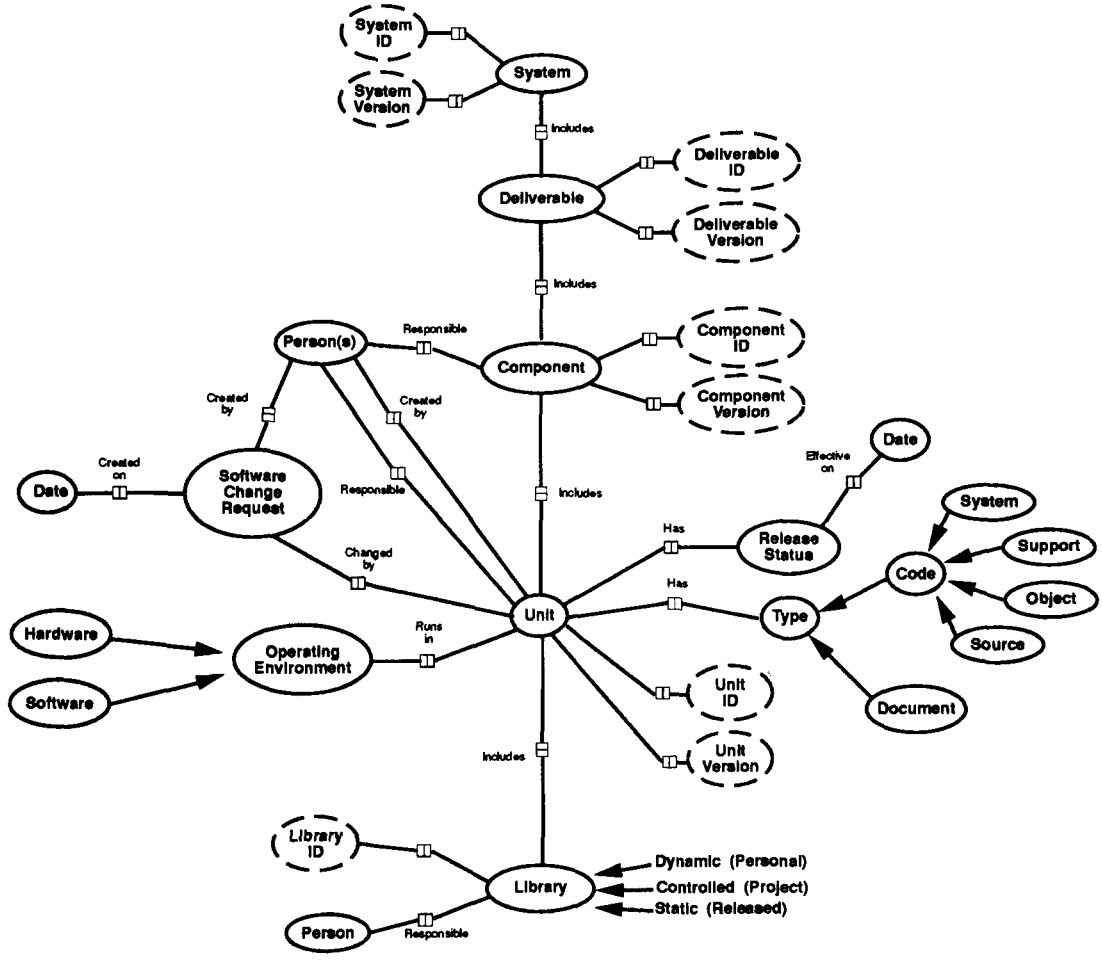


Figure 5-2. Information Model

A unit is included in a component, which in turn is included in a deliverable. One or more deliverables may be included in a system. If a unit is part of a controlled or static (project or released) library, then a change request must be generated and approved before it can be changed. A change request has an identifier, a creator, a date, is related to one or more units, and has a change review board that must approve it.

5.4 Pitfalls

There are many pitfalls in establishing software configuration management in an organization and in automating it once it has been established. Obviously, the risks are greater going from no SCM to an automated SCM, than in first going to a manual system, which is later automated.

Some SCM pitfalls occur at the project level, while others occur at the organizational level, *i.e.*, across projects.

One pitfall is to create an imbalance of cost to benefit. Sometimes there is a discrepancy in that one group gets the benefits, while a different group primarily sees the costs. To avoid this, SCM should not burden the developers more than it benefits them. This may be difficult to assess early in the development process since it provides a longer term benefit, not an immediate benefit for them.

Another problem is matching the level of configuration management to the type of project. Projects can be classified along two dimensions, by size (small, medium, and large) and by how well they are structured. For well-behaved (*i.e.*, the users know what they want and their requirements are stable), small projects, SCM is of less concern. As project size increases, so does the benefit of software configuration management as a coordinating and control tool. As the lack of structure increases so does the benefit of SCM. Because unstructured projects have many changes during the development process, SCM is essential to coordinate and control these changes. As project size increases (given the same amount of structure), SCM benefits the developers by easing their coordination and communications problems. As the amount of structure decreases (given the same size), SCM benefits communications and coordination between the developers and the users. Finally, SCM also can provide benefit if the project is working on a common problem or where part of the software can solve a common problem, because SCM makes it easier to develop and find reusable code.

SCM for a project should be started when the project is begun because it is very difficult to retrofit it into an existing project, especially a very large one.

For reusability there is also an issue of granularity (*i.e.*, which software entities to identify, control, and track). All projects within an organization should use a common taxonomy for software entities, such as the one described in Section 5.3. Small and/or early projects (or manual ones without automated tools) may only track the higher level entities rather than all of the entities in the taxonomy. However, they should also use the same taxonomy.

In summary, automated tools can provide support for SCM, but they are not the critical factor. They can reduce the administrative detail and provide easier access to the data, but without the commitment to do the human activities of SCM, the automated tools will not make the effort a success.

Appendix A

References

[BER80] **Bersoff, Edward H., Vilas D. Henderson, Stanley G. Siegel,** *Software Configuration Management: An Investment in Product Integrity,* Prentice Hall Inc., Englewood Cliffs, NJ, 1980.

[BOE86] **Boehm, Barry W.,** "A Spiral Model of Development and Enhancement," ACM SIGSOFT Software Engineering Notes, Vol. 11, No. 4, Aug 1986, pp. 14-24.

[EP016] **Engineering Procedure, EP401016,** "Identification Marking," Issue AA.

[EP032] **Engineering Procedure, EP401032,** "Product Change Control," Issue H.

[EP033] **Engineering Procedure, EP401033,** "Revising Drawings and Part Numbers to Define Product Changes," Issue P.

[EP035] **Engineering Procedure, EP401035,** "Sandia and Production Agency Acceptance Equipment Interfaces," Issue K.

[EP040] **Engineering Procedure, EP401040,** "Drawing System," Issue H.

[EP043] **Engineering Procedure, EP401043,** "Acceptance Equipment Program Definition," Issue F.

[EP044] **Engineering Procedure, EP401044,** "Engineering Release System," Issue J.

[EP045] **Engineering Procedure, EP401045,** "Definition of Computer Software Configuration Items," Issue C.

[EP054] **Engineering Procedure, EP401054,** "Nine-Digit Part Number System," Issue C.

[EVA83] **Evans, Michael W., Pamela Piazza, James B. Dolkas,** *Principles of Productive Management,* John Wiley & Sons, New York, 1983.

[EVA87] **Evans, Michael W., John Marciniak,** *Software Quality Assurance and Management,* John Wiley & Sons, New York, 1987.

[IEE87] **The Institute of Electrical and Electronics Engineers, Inc.,** IEEE Guide to Software Configuration Management, ANSI/IEEE Std 1042-1987.

[IEE89] **The Institute of Electrical and Electronics Engineers, Inc.,** IEEE Standard for Software Quality Assurance Plans, ANSI/IEEE Std 730-1989.

[IEE90-a] **The Institute of Electrical and Electronics Engineers, Inc.**, IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Std 610.12-1990.

[IEE90-b] **The Institute of Electrical and Electronics Engineers, Inc.**, IEEE Standard for Software Configuration Management Plans, ANSI/IEEE Std 828-1990.

[IEE91] **The Institute of Electrical and Electronics Engineers, Inc.**, IEEE Software Engineering Standards, Spring 1991 Edition, 1991.

[SSGv1] ***Sandia Software Guidelines,*** Volume 1, "Software Qualtiy Planning," SAND85-2344, Sandia National Laboratories, Albuquerque, NM, Aug 1987.

[SSGv2] ***Sandia Software Guidelines,*** Volume 2, "Documentation," SAND8545, Sandia National Laboratories, Albuquerque, NM, Planned for 1993.

[SSGv3] ***Sandia Software Guidelines,*** Volume 3, "Standards, Practices, and Conventions," SAND85-2346, Sandia National Laboratories, Albuquerque, NM, Jul 1986.

[SSGv4] ***Sandia Software Guidelines,*** Volume 4, "Configuration Management," Sandia National Laboratories, Albuquerque, NM, Jun 1992

[SSGv5] ***Sandia Software Guidelines,*** Volume 5, "Tools, Techniques, and Methodologies," SAND85-2348, Sandia National Laboratories, Albuquerque, NM, Jul 1989.

Appendix B

Glossary and Acronyms

ACO: Advance Change Order

ASCII: American Standard Code for Information Interchange

Baseline: A baseline is the documented identification of a software product Configuration Item (CI) — its code and all its related documentation at some specific point in time. It is the basis for all Software Configuration Management (SCM) activities.

CASE: Computer-Aided Software Engineering

CCB: Configuration Control Board. A CCB is a group of people responsible for evaluating, and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes. A CCB is sometimes referred to as a Change Control Board.

CER: Complete Engineering Release

CI: Configuration Item. A collection of (software) entities (or components) treated as a single element for the purpose of configuration management. Configuration Items can vary in size, complexity, and type. A CI may also be called a Computer Software Configuration Item (CSCI), Computer Program CI (CPCI), or (software) system segment.

CM: Configuration Management

CMP: Code Maintenance Plan

CPC: Computer Program Component, same as Computer Software Component (CSC)

CPCI: Computer Program Configuration Item, same as Computer Software Configuration Item (CSCI)

CSA: Configuration Status Accounting

CSC: Computer Software Component. A distinct part of a software Configuration Item (CI). A software component may also be called a Computer Software Component (CSC) or Computer Program Component (CPC). CSCs may also be further decomposed into other CSCs or individual units. If a large analysis program is called out as a CI, some of the CSCs could be a requirements document or major groupings of the software modules.

CSCI: Computer Software Configuration Item

CSF: Common File System

DBMS: Data Base Management System

DIR: Defect Investigation Report

DMS: Design Management System

EER: Engineering Evaluation Release

EP: Engineering Procedure

FCA: Functional Configuration Audit

FCO: Final Change Order

FTE: Full-Time Equivalent

HOL: Higher Order Language

IEEE: Institute of Electrical and Electronics Engineers

IFS: Integrated Files Store

NSS: Network Storage Service

LAN: Local Area Network

ML: Materials List

OTS: Off-the-Shelf

PCA: Physical Configuration Audit

Personal Library: The Personal Library (also termed the dynamic or programmer's library) is used for holding newly created or modified software entities. This library constitutes a software developer's workspace for writing new code or documentation, and may take any form suitable to the developer's needs, but should have a degree of order to it that allows the status of its entities to be determined easily. Each software developer may have a Personal Library from which project entities can be linked and/or copied. Alternately, each small team may have a Personal Library assigned to the team with lower level personal libraries assigned to each individual, especially in a Local Area Network (LAN) or mainframe environment. Access to the Personal Library is controlled by, and usually limited to, the software developer.

Project Library: The Project Library (also termed the controlled or "master" library) is a library used for managing the current internal developmental baselines and for controlling changes made to them. This library represents the latest internally-approved version of the software product being developed. Changes to software entities in the Project Library should have gone through formal approval procedures established in a configuration management plan. Code in this library should

have been tested sufficiently to ensure that it is ready for integration. Copies may be freely made for use by the software developers and others, but changes must be strictly controlled and documented in order to ascertain at any given point its exact configuration. Even for simple projects in which there is only one code developer, there still should be functional separation between the Personal Library and the Project Library.

PROM: Programmable Read-Only Memory

Promotion: A promotion is an action taken with a software component to increase the level of authority needed to approve changes to it. For example, a top-level software design description is promoted or moved into the Project Library where all developers can view, but not modify it without proper authority. This allows the developers to work on issues that may concern detailed designs and implementation.

QA: Quality Assurance

QE: Qualification Evaluation

QER: Qualification Evaluation Release

RAN: Restricted Access Network

Release: A release is a copy of the software CI or CSC that is turned over to the customer or user. It is a promotion of that CI or CSC outside of the development organization.

Release Library: The Release Library (also termed the static or "software repository" library) is a library used to archive the various baselines (versions) released for general use. This library is never changed (except to add a new version), since it must be able to duplicate results from software that has been released for operational use by other organizations. Access should be limited to "read only" for the purpose of making copies. The Sandia Drawing System provides a Release Library capability required for War Reserve (WR) projects and is useful for most other projects involving formal software deliverables.

Revision: A revision is a formal change to a software CI or CSC that does not alter its documented functional or performance capabilities. An example of this is when code is changed to correct a fault.

SCCB: Software Configuration Control Board. A Software Configuration Control Board is a group of individuals who oversee the software change process, with ultimate authority for approving a change and promoting a software entity from one library to another. The individuals may be from the project, related organizations and management levels, the customer, or some combination. During the development process, the SCCB controls promotions into the Project Library from the Personal Library and changes to the products in the Project Library. During the support phase, the project CCB and SCCB provide the authority to make changes to products already promoted to the Release Library, and to promote software products from the Project Library to the Release Library. An SCCB may also be referred to as a Software Change Control Board.

SCM: Software Configuration Management

SCMP: Software Configuration Management Plan

SCR: Software Change Request

SDD: Software Design Description

SDP: Software Development Plan

SLOC: Source Lines of Code

SNL: Sandia National Laboratories

SPMP: Software Project Management Plan

SQA: Software Quality Assurance

SQAP: Software Quality Assurance Plan

SQP: Software Quality Plan

SRS: Software Requirements Specification

Version: A version is a software CI or CSC with a defined set of capabilities. A new version is a variation of the previous version in that it has a change in its functionality or performance characteristics. An example of this is a change to the software to generate a different output.

WR: War Reserve

Appendix C

Organizational SCMP Thematic

Note: Italics type is instructional information that should be deleted when a section is filled in with the necessary information for the project. Times Roman text can be revised as necessary and become part of the final document.

1. INTRODUCTION

1.1 Purpose

The purpose of this plan is to describe the software configuration management policies, organization, activities, and relationships within the Sandia National Laboratories (SNL) Component Development Vice Presidency (N000). All War Reserve (WR) and WR-like software developed within the directorates and divisions of N000 shall* implement software configuration management in accordance with this plan. This plan will provide a foundation for application of consistent software configuration management practices across projects critical to the SNL mission.

1.2 Scope

This plan shall apply to WR and WR-like software developed by any projects within the Component Development Vice Presidency (N000). The management and organizational relationships are included in this plan along with guidance for development of specific project software configuration management plans that will be consistent with this general organizational plan. The applicable policies and procedures to be applied to project software configuration management are described. It is recommended that projects involving non-WR software develop a project-specific configuration management plan in accordance with this general organizational plan.

1.3 Definitions and Acronyms

ANSI - American National Standards Institute
AR - Automatic Reserve
CCB - Configuration Control Board
DoD - Department of Defense
EP - Engineering Procedure
IAW - In accordance with
IEEE - Institute of Electrical and Electronics Engineers
SCM - Software Configuration Management
SCMP - Software Configuration Management Plan
SNL - Sandia National Laboratories
SQAP - Software Quality Assurance Plan
SSG - Sandia Software Guidelines

---

* *Note that the requirements of a plan are indicated by the use of "shall" and sometimes "will." Options are identified by "should" or other less directive words.*

STD - Standard
WR - War Reserve

1.4 References.

[1] Sandia National Laboratories Quality Plan
[2] Sandia Software Guidelines, Vols 1, 2, 3,4, 5
[3] SNL Software Development Policy and Recommended Software Development Practices, L. D. Bertholf memorandum, November 19, 1990
[4] ANSI/IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology
[5] ANSI/IEEE Std 828-1990, IEEE Standard for Software Configuration Management Plans
[6] ANSI/IEEE Std 1042-1987, IEEE Guide to Software Configuration Management
[7] DOD-STD-2167A, Military Standard, Defense System Software Development, Chapter 4.5, Software Configuration Management
[8] EP401016, "Identification Marking"
[9] EP401032, "Product Change Control"
[10] EP401033, "Revising Drawings and Part Numbers to Define Product Changes"
[11] EP401035, "Sandia and Production Agency Acceptance Equipment Interfaces"
[12] EP401040, "Drawing System"
[13] EP401043, "Acceptance Equipment Program Definition"
[14] EP401044, "Engineering Release System."
[15] EP401045, "Definition of Computer Software Configuration Items"
[16] EP401054, "Nine-Digit Part Number System"

2. SCM MANAGEMENT

2.1 Organization

2.1.1 Policy

Each project involving WR or WR-like software shall develop a Software Configuration Management Plan (SCMP) in accordance with the Sandia Software Guidelines (SSG, reference [2]) and this organizational plan. Change control will be the responsibility of the individual project as described in the SCMP. The development and support change responsibilities will be defined in the SCMP. The SCMP software configuration review board will provide status to the general organization software configuration control board as described in subsection 3.3 of this plan. The software product will be controlled in accordance with EP401045 and its related Engineering Procedures.

Note that the Organizational SCMP could be a part of an Organizational Software Quality Assurance Plan. Likewise, the Project-Specific SCMP might be a part of a Project-Specific SQAP. These options should be part of the policy statement as appropriate.

2.1.2 Organization and Relationships

Include organization chart of VP N000 with its key departments and auxiliary interfaces with other VP organizational entities such as systems engineering, quality assurance, and drawing system.

Indicate the functional roles of systems engineering, component engineering, software development/support, and quality assurance.

Describe the general organizational relationship to the project specific configuration management organization, specifically via the control board interface with the software configuration review board.

2.2 SCM Responsibilities

2.2.1 General VP N000 SCM Responsibilities

Indicate the responsibilities of the Organizational SCCB. Membership might include Directors, Supervisors, and Software Project Leaders. Perhaps a representative from Software Quality and Reliability Engineering could be a member to provide general SNL software guidance and some level of independence. Note that the software functions of the Organizational SCCB might be a part of a more general Organizational CCB that considers the overall system configuration issues for systems designed/developed by VP N000 personnel.

Indicate the major functions, interfaces, inputs/outputs of the SCCB. Primary functions are major project release decisions and major project rework decisions. Interfaces include customer and internal VP directorate / division / project. Inputs are release packages and accompanying analysis data; outputs are decisions (release, rework, kill).

Indicate that the Organizational SCM Library will be the Sandia Drawing System as defined in References [8] - [16]. Indicate that the Project-Specific Librarian will release project software products to the Sandia Drawing System through the Project-Specific SCCB after concurrence by the Organizational SCCB.

Organizational data bases reference information in project-specific data bases as well as summary information across all projects.

2.2.2 Project-Specific SCM Responsibilities

Indicate the SCM responsibilities of the project and the relationship to the Organizational SCCB. Include a figure. Responsibilities should include Project-Specific Software Librarian and Project-Specific SCCB. Depending upon the project size, the Project SCCB composition could vary considerably.

Major responsibilities are to communicate final project release package to the Organizational SCCB, and respond to requests for project release rework. Responses to questions of possible software project adaptation (e.g., reuse) for new questions would be processed through the Organizational SCCB to the Project-Specific SCCB.

Other Project-Specific SCM responsibilities are described in the IEEE Standard and Guidelines (references [5] and [6]). This Organizational SCMP should require adherence to the Sandia Drawing System configuration identification and control procedures as described in the Engineering Procedures, in particular EP401045.

2.3 Applicable Policies, Directives, and Procedures

Reference the SNL Software Quality Plan, VP2000 Software Development Policies, SSGs, and EPs. Describe briefly the significance of why each referenced document applies to this organization.

3. SCM ACTIVITIES

3.1 Identification

Project-specific SCMPs shall define the implementation of EP401045 for the project along with any lower level identification details. Identification of developmental baselines and product baseline updates will be described at the lower level.

EP401045 defines the general identification scheme.

Typical baselines will include the functional, allocated, and product.

Software product elements to be identified shall include

(1) application software products
(2) system support and utility software
(3) maintenance and test software
(4) equipment required for development/test/support

3.2 Control.

Processing of changes will generally be handled in accordance with the Project-Specific SCMP. Changes that might affect multiple software products and/or weapons systems will be coordinated through the Organizational SCCB.

Two-tiered level of change authority control. Organizational SCCB supported by Project-Specific SCCBs. Indicate the authority relationships, control flow, and associated information flow.

Support software, vendor-provided software, and any other software required to develop/support the primary software products will be controlled in accordance with (IAW) the Project-Specific SCMP.

3.3 Status Accounting

Status of initial versions and updates to functional, allocated, or product baselines will be reported to the Organizational SCCB through the Project-Specific SCCBs. Status of released products will be available through the Sandia Drawing System.

A standard data base of configuration status and changes for all software projects will be supported. Each Project-Specific SCMP shall support report inputs to this data base for changes and status.

3.4 Audits and Reviews

The Organizational SCCB will review each project's SCMP, status of controlled baselines IAW that project's SCMP, and issues brought to the attention of the SCCB by individual SCMPs. The Organizational SCCB will meet at least twice a year to review all software projects to ensure compliance with this plan.

3.5 Interface Control

Indicate the general interface of the Organizational SCM to overall system configuration management and the relationships to other functional organizational elements such as systems engineering, software development, software support, and quality assurance. The theme should be an integrated function along the lines of Appendix D SCMP in IEEE Std 1042-1987 (reference [6]).

3.6 Subcontractor/Vendor Control

Any subcontractor software shall be controlled IAW the subcontractor's SCMP. The subcontractor's concept will be described in the Project-Specific SCMP, and shall be consistent with this Organizational SCMP, the Project-Specific SCMP, and the SSG. The subcontractor's SCMP shall be reviewed by the project personnel.

Any vendor-supplied software will be described in the Project-Specific SCMP.

Control of updates to this software, response to software problems, and issues such as availability of the software if the vendor business status changes (i.e., absorbed by another company, goes out of business).

4. Schedules

4.1 Organizational SCM Plan Implementation

Indicate THIS plan's implementation considerations. In particular, THIS plan is implemented by the creation of the Organizational SCCB function and Project-Specific SCMPs in accordance with THIS plan.

Indicate the intent to allow projects to tailor Project-Specific SCMPs to the appropriate level of detail per the project needs. The Organizational SCCB will review and approve the project-specific plans.

5. Resources

5.1 Data

The Organizational SCCB will use a standard data base for tracking status and changes to controlled software product baselines. This data base will be IAW the Sandia Drawing System.

An internal data base will provide the mechanism for collecting and retaining records for all projects that pertain to SCCB activity, status, release reports, change requests, and any metrics that indicate configuration management process improvement.

Lower-level records collection and retention necessary for software development and support will be documented in the Project-Specific SCMP.

5.2 Tools, Techniques, and Methodologies

Project-Specific SCMPs will describe specific tools, techniques, and methodologies to be applied to the project software IAW the SSGs. It is the goal to standardize software configuration management tools, techniques, and methodologies as much as is practical across software projects.

Typical classes of tools required include project library management system, code management system, systems/software change request tracking system, software change authorization system, and status accounting report generation system.

The project library management system provides the capability to organize the project development/support environment into software libraries. Personal, project, and release libraries are separated and controlled. Access to the libraries is controlled, and the personnel's interface to the development/support environment is defined.

The code management system provides context-sensitive editors, software source version control, software load build capabilities, and compatibility between source and object versions.

The change request tracking system maintains information on each change request and the status of each change request.

The software change authorization system provides the capability to link authorization information with each change request and releases of software products.

The status accounting report generation system provides reports summarizing the status (accepted, deferred, rejected) of all changes requests, current baseline development/update status, and schedule data for future activities.

Each Project-Specific SCMP will describe the use of software libraries for the evolution of baselines for development and support. Personal, project, and release libraries will be supported. Their use for each project will be described in the Project-Specific SCMP.

5.3 Personnel

Identify personnel requirements to support configuration management functions at the VP level.

5.4 Training

Identify any initial and recurrent training required for personnel to function in software configuration management positions or to utilize any of the identified tools used to control software products.

6. Plan Maintenance

Ownership of this Organizational SCMP resides in the NN00 Directorate's Quality Coordinator for Software Development. It is reviewed at least annually for update consideration. Suggestions and comments should be directed to the Quality Coordinator for Software Development, NN00.

Changes to this plan are reviewed and approved by the N000 SCCB.

Appendix D
Sample Project Software Configuration Management Plan:
Small to Medium Weapons Application

Software Configuration Management Plan

for the

MC0001

FB112 Launch Controller

February 8, 1997

Prepared by: _____ / /97 O. K. Kanagal, 5184

Reviewed by: _____ / /97 E. T. Summer, 0326

Approved by: _____ / /97 M. A. Baca, 0326

_____ / /97 X. I. Ting, 5184

_____ / /97 P. D. Quick, 2368

_____ / /97 U. C. Taylor, 2317

Sandia National Laboratories

Albuquerque, NM 87185

1. Introduction

1.1 System Overview. The MC0001 Launch Controller is being designed for the FB112 Single Stage to Orbit Aerospace Vehicle. The MC0001 is being designed to be compatible with the W144 Photon Torpedo and W126 Particle Beam Generators, as well as conventional nuclear devices that the FB112 is capable of carrying. The MC0001 employs two microcontrollers containing 4K bytes each of Programmable Read-Only Memory (PROM) and one microprocessor containing 8K bytes of PROM. The two microcontrollers are utilized in the interface control subsystem and the microprocessor is utilized in the launch processing and monitoring subsystem. The interface control software is being designed to be implemented on Sandia SA348x microcontrollers, the functional equivalent of the INTEL 80C532 microcontroller. The software for launch processing and monitoring is being designed to be implemented on a Sandia SA352x, the functional equivalent of the INTEL 80786 microprocessor. The software is being developed on SUN Workstations operating in a Local Area Network (LAN). The software is being written in C.

1.2 Purpose. This software configuration management plan (SCMP) specifies the requirements and procedures necessary for the configuration management activities of the MC0001 project. It defines the methods to be used for identifying software entities, implementing and controlling changes, and recording and reporting the implementation status of software configuration items (CIs).

1.3 Scope. The organizations involved in this project are identified in Figure 1. This plan applies to all phases of the software development life cycle including postdeployment support if necessary.

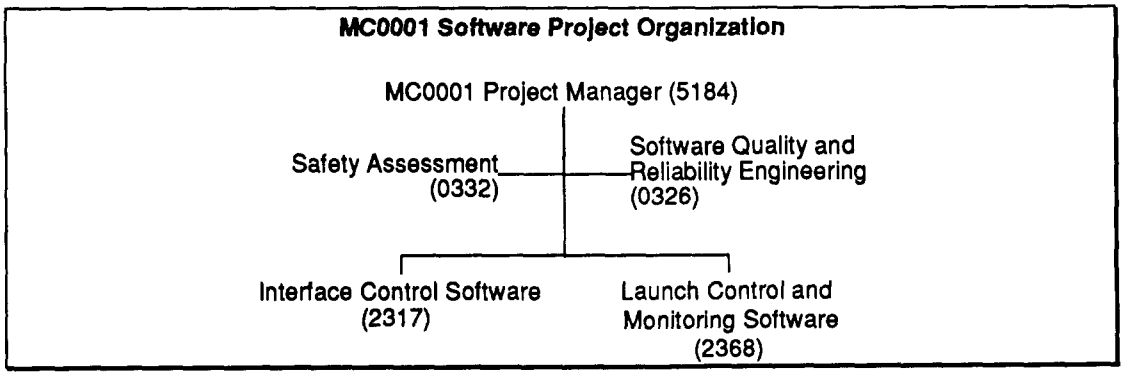


Figure 1

This plan applies to all software and associated documentation used in the production of computer programs produced for the MC0001 project. It includes the software development activities that support the iteration of prototype builds of the MC0001 hardware. Software CIs controlled by this plan include

MC0001 Beam Weapon Interface Program
MC0001 Nuclear Weapon Interface Program
MC0001 Launch Control and Monitoring Program

MC0001 Diagnostic and Test Software
MC0001 Emulation Program

1.4 Definitions and Acronyms. The definitions used in this plan conform to Sandia Software Guidelines (SSGs), Volumes 1, 3, and 4. Other definitions will conform to those found in ANSI/IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology. Unique definitions used in this plan include

Interface Control. The process of (1) identifying all functional and physical characteristics relevant to the interfacing of two or more CIs provided by one or more organizations, and (2) ensuring that proposed changes to these characteristics are evaluated and approved prior to implementation.

The following acronyms are referred to within the text of this plan

AFSC	Air Force Systems Command
CASE	Computer-Aided Software Engineering
CCB	Configuration Control Board
SR	Software Requirements Drawing
CD	System Compatibility Drawing
CI	Configuration Item
CM	Configuration Management
LAN	Local Area Network
NSA	National Security Agency
PROM	Programmable Read Only Memory
QE	Qualification Evaluation
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SPACECOM	Space Command
SQE	Software Quality Engineer
UART	Universal Asynchronous Receiver/Transmitter
USAF	United States Air Force

References. The documents listed here will be considered when applying this plan (latest revisions apply):

[1] Sandia National Laboratories Quality Plan

[2] *Sandia Software Guidelines,* Vol 1, "Software Quality Planning"; Vol 2, "Documentation"; Vol 3, "Standards, Practices, and Conventions"; and Vol 4, "Software Configuration Management"

[3] ANSI/IEEE Std 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology"

[4] ANSI/IEEE Std 828-1990, "IEEE Standard for Software Configuration Management Plans"

[5] ANSI/IEEE Std 1042-1987, "IEEE Guide to Software Configuration Management"

[6] NSA Software Product Standards Manual, Chapter 2.3, Software Configuration Management

[7] DOD-STD-2167A, Military Standard, Defense System Software Development, Chapter 4.5, Software Configuration Management

[8] Statement of Work, FB112 Launch Controller, Order No. 6Y450-96, USAF/AFSC, Wright Patterson AFB, OH, April 1, 1996

[9] Sandia/Rockwell ISR HD4901, Interface Control - Avionics to Launch Controller - Programming and Timing/Electrical

[10] SR999996, Interface Control Document for the MC0001 Launch Controller, Issue B

2. Software Configuration Management

2.1 Organization. Special Systems Development (5184) has overall responsibility for the MC0001 development project. The authority for software configuration management (SCM) and implementing the SCMP is assigned to Digital Subsystem Software II (2317) for the Interface Control Software and to Advanced Firing Set II (2368) for the Launch Control and Monitoring Software, the Diagnostic and Test Software, and the Emulation Program. Organizations 2317 and 2368 will each appoint an SCM coordinator for their respective software responsibilities. The SCM coordinators will co-chair the MC0001 configuration control board (CCB). Organization 5184 and Software Quality and Reliability Engineering (0344) will each assist in the management of the SCMP and provide representation to the MC0001 CCB. Safety Assessment Technologies (0332) will also provide representation to the MC0001 CCB. Hereafter, the MC0001 CCB will be referred to as the CCB.

2.2 SCM Responsibilities. Organizations 2317 and 2368 responsibilities for their respective development assignments include configuration identification, configuration control, status accounting, co-chairing the CCB, establishment and maintenance of developmental baselines, and participating in reviews and inspections. Organization 5184 responsibilities include liaison with USAF/AFSC, USAF/SPACECOM, and Rockwell International; systems integration; and providing representation to the CCB. Organization 0344 is responsible for maintenance of the SCMP, chairing formal reviews and audits, consulting with Organizations 2317, 2368, and 5184 on software Quality Assurance (SQA) matters, and providing representation to the CCB.

2.2.1 Configuration Identification. Configuration identification is applied to all MC0001 software entities, both code and associated documentation. Each software entity created for the MC0001 will be assigned a configuration identification alphanumeric before it is begun. Software entities that are source code modules will be combined into a single configuration item (CI) for each major function when baselined and promoted to the Project Library. This will also be the CI that is released to the drawing system (Release Library). Documents are established as CIs after they are baselined.

2.2.1.1 Baselines. Baselines are established for the control of requirements, design, and product changes and are time phased to the development of the MC0001. Baselines are established by the organizations responsible for their respective CIs at the completion of formal inspections and approval of those inspections by the MC0001 qualification evaluation (QE) team. Baselines defined for the MC0001 include

[1] Functional baseline. In addition to the above, the functional baseline is established by customer approval of the MC0001 system requirements or system compatibility drawing (CD drawing).

[2] Allocated baseline. The allocated baseline is established when the software requirements specification (SR drawing) has been inspected and approved as described above.

[3] Product baseline. The product baseline is established when the source code has been inspected and approved as described above and released into the Sandia drawing system.

2.2.1.2 Libraries. Throughout the development cycle, each software entity is promoted to higher levels of control as its existence becomes increasingly defined. Libraries fall into three categories

[1] Personal Library. Maintained by the designer/programmer in password write-protected files.

[2] Project Library. Maintained by the designated configuration manager in password write-protected files.

[3] Release Library. Maintained in the drawing system.

2.2.2 Configuration Control. All software entities are maintained by the controlling authority for the respective library in which the entity resides. For entities in the Personal Library, the designer/programmer has individual responsibility to control changes. Inspected and baselined entities in the Project Library are changed via the Software Change Request (SCR) system by a designated designer/programmer with the approval of the CCB. In addition to the SCR, changes to CIs in the Release Library require an Advance Change Order (ACO) or Final Change Order (FCO) issued by the CCB or the caretaker configuration manager if the project has been completed and closed out. Note that initial release (or promotion) to the drawing system (Release Library) requires a Complete Engineering Release (CER) to be issued against the respective CIs. Baselined documentation and software entities will be managed and accounted for with Amplify Control, a commercially available computer aided software engineering CASE tool operating on the Sun workstation.

2.2.3 Status Accounting and Audits. Reports on the status of software or documentation in personal libraries or the project library may be generated at any time using the CASE tool. Audits will be performed at the request of the USAF through the project manager.

2.2.4 Configuration Control Board. The makeup of the CCB is described in 2.1 above. The CCB will meet as required to review change requests to software or documentation in the project or release libraries. The CCB will issue one of the following decisions: approved, disapproved, or

tabled. Tabled requests will create an action item to resolve the issue(s) causing a tabled decision and set a date to reconsider that item.

2.3 Applicable Policies, Directives, and Procedures. In addition to this SCMP the following documents apply for the life of this project:

Ref. [2]. *Sandia Software Guidelines*, Vol 3, "Standards, Practices, and Conventions"

Ref. [6]. NSA Software Products Standards Manual, Chapter 2.3

Ref. [8]. Statement of Work, FB112 Launch Controller

3. SCM Activities

3.1 Configuration Identification. A six-digit part number from the Sandia drawing system with prefixes selected from Table 1 and a three-digit correlation suffix shall be used to identify each CI. A software CI includes requirements specifications, design and test documentation, and compiled source and object (or machine executable) code. Configuration identification of computer programs and documentation during the development effort consists of established baselines and libraries that are time phased to the development schedule of the MC0001. Until a software CI is promoted to the Release Library, it will be identified as Issue 0 as it remains in the Personal or Project Library. Once in the Release library, the CI will be established as Issue A and subsequent changes to a baseline will be identified as Issue B, C, etc. Also, a page change table will be in the beginning of each baselined document. Significant changes to any CI will be denoted by incrementing the correlation suffix, resulting in a different version of that CI. Modules (subroutines) will begin maintaining a revision history using a "dot" and two digit notation (e.g., .01, .02) added to the module nomenclature as they are established in the Personal Library.

3.2 Configuration Control. Software documentation as well as software modules maintained at the Personal and Project Library level will be physically controlled using Amplify Control. Software modules and documentation in the Personal Library will be change controlled by the responsible developer. Modules and documentation in the Project Library will be change controlled by the CCB. All software entities released to the drawing system (Release Library) can only be changed by ACO/FCO action after approval by the CCB.

3.2.1 Function of the Configuration Control Board. The CCB acts as the governing authority to ensure that proposed changes to formally identified CIs and any composite entities comply with approved specifications and designs, and it evaluates the impact of those proposed changes on existing software. After the code has been inspected and promoted to the Project Library, the CCB may delegate this authority to a subboard consisting of two or more CCB members for change requests involving coding during integration and system testing. In this case, the CCB will review the changes prior to promoting the source and object code to the Release Library and retain the authority to veto any change.

Table 1.

Configuration Identification Prefixes	
SR	Software Requirements Specification
SD	Software Design Description
GD	Software Design Description (Graphical)
PD	Software Program Description (Compiled Source Code Listing)
AM	Magnetic Media containing Source Code and any other code (compiler, assembler/linker or make files) required to construct the binary object (machine executable) code
TK	Software Test Plan, Test Cases and Test Results
AT	Magnetic Media containing Object (machine executable) Code and other files required to insert the program into PROMs

3.2.2 Software Change Request/Change Authorization. An SCR will be used to track all changes to software code and documentation that reside in the Project or Release Libraries (baselined software). The SCR form (Attachment 1) will contain a narrative description of the change or problem, information to identify the source of the report, and some basic information to aid in evaluating the report. It will be submitted to the appropriate SCM authority, depending on which software program it pertains to, and will be sequentially numbered using an automatic tracking system. Anyone associated with the MC0001 project may submit an SCR, but usually an SCR is submitted by a member of the development team. SCRs will be evaluated by the CCB (or CCB subboard) and approved or disapproved for implementation. An SCR is closed when (1) Integration testing has shown that the changes have been correctly made, (2) No unexpected side effects have resulted from the change, and (3) Documentation has been updated and reviewed.

3.3 Configuration Status Accounting. Amplify Control provides an on-line inquiry capability and user-specified report generation. Amplify Control will also provide a version description document and software configuration tree. These can be generated on demand, but will be produced quarterly and retained by the CCB. The SCR tracking software is capable of generating reports on the status of SCR as well as the entire SCR data base. Status reports will be produced monthly and retained by the CCB.

3.4 Audits and Reviews. No audits are scheduled for MC0001 software. However all software entities are inspected or reviewed by the QE team prior to their release to the drawing system (Release Library). Details of these inspections and reviews are documented along with other qualification activities in the Engineering Evaluation Releases (EERs) and Qualification Evaluation Releases (QERs) for the MC0001.

All product definition drawings including test planning and test results produced for the MC0001 is retained and safeguarded in the Sandia Drawing System. Copies of SCM status reports, CCB activities, and SCR will be maintained in Organization 2317 in an MC0001 project file for a period of one year after the completion of the MC0001.

3.5 Interface Control. Organization 2317 is responsible for interface control between the SA348x microcontrollers and the SA352x microprocessor. Organization 5184 is responsible for the MC0001 external interface with the FB112. Note: Hardware/software interface control is documented explicitly in the INTEL Microcontroller and Microprocessor Handbooks.

3.6 Supplier Control. Vendor-supplied software consists of compilers, emulators, and CASE tools. Although good business practice SCM is expected and notification of new version releases are provided, no review of vendor SCM planning is performed.

4. Schedules

SCMP Implementation. This SCMP is implemented as soon as it is approved by the cover page signatories. No baselines may be established prior to implementation of this SCMP. The CCB is established at the time of SCMP approval. Baselines are described in 2.2.1.1 above.

5. Resources

5.1 Tools, Techniques, and Methodologies. The primary tool for physical control of software in personal and project libraries is Amplify Control, an automated configuration management tool that provides source code control, structure charts, documentation preparation, password control, and other configuration management functions. The engineering release system is used to control all changes to CIs in the Release Library (drawing system). The SCR is the primary technique to identify, track, and manage changes to software in the Project and Release Libraries. The SCR system is a combination of automated identification and tracking of software changes, and the human process of evaluating and approving or disapproving those changes.

5.2 Personnel and Training. All personnel working associated with the development of the MC0001 have or will complete INTEC course CS842, Software Configuration Management. No additional training is required for developers to utilize Amplify Control.

6. Plan Maintenance

This plan is owned by Organization 2317. It is reviewed semiannually by the MC0001 CCB. Suggestions or comments should be sent to MC0001 SCM coordinator, 2317. All changes to this plan shall be approved by the MC0001 CCB.

Attachment 1
Software Change Request

SCR Number:_____

1. Submitted by:_____ Date:__/__/__

2. Software Program/Document Name:_____

   Configuration Identification/Part Number:_____

   Revision/Issue:_____

3. SCR Type: [ ] (1 - Development 2 - Problem 3 - Enhancement)

4. Short Task Description:

5. Detailed Description:

6. Submitter's Priority: [ ]
   (1-Critical 2 - Very Important 3 - Important 4 - Inconvenient 5 - Interesting)

7. CCB Action:_____ CCB Priority: [ ]

8. Assigned to:_____ Suspense Date:__/__/__

9. Solution Comments:

10. Software Programs Affected:

11. SCM Approval:_____Date:__/__/__

12. Closed by:_____Date:__/__/__

APPENDIX E
Sample Project Software Configuration Management Plan: Large, Non-Weapons Research Application

MELCOR Software Configuration Management Plan

October 4, 1990

Randall M. Summers
Thermal/Hydraulic Analysis
Division 6418
Sandia National Laboratories
Albuquerque, NM 87185

Contributions by:
Randall K. Cole, Jr.
Stephen W. Webb

1. INTRODUCTION

1.1 Purpose

The MELCOR Software Configuration Management (SCM) Plan describes the environment and procedures for development and maintenance of the MELCOR Code System (hereafter usually referred to as simply "MELCOR"). It is intended that this plan conform to requirements laid out in ANSI/IEEE Std 828-1983, "IEEE Standard for Software Configuration Management Plans," as well as the Quality Assurance Program Plan for the Reactor Systems Safety Department 6410 at Sandia National Laboratories.

1.2 Scope

For the purpose of this SCM Plan, MELCOR includes all program libraries, FORTRAN source code, object libraries, executables, and documentation for the MELGEN, MELCOR, and MELPLT programs, which comprise the MELCOR code system. MELCOR contains a very large number of subroutines (well over a thousand), which are grouped into packages according to the phenomena they model or the function they perform. Compilable FORTRAN source amounts to over a quarter million lines of code. Documentation is subdivided into individual Users' Guides, Reference Manuals, and Programmers' Guides for each MELCOR package. MELCOR is developed and maintained principally on a VAX 8700 computer running the VAX/VMS operating system; methods and procedures for MELCOR SCM are determined in part by this computing environment. Approximately 200,000 512-byte VAX blocks are required for configuration management of MELCOR.

The primary configuration items for MELCOR are the program libraries, which are created and modified for each of the various MELCOR packages by a Sandia-developed software product called Code Maintenance Package, or CMP. (CMP is a set of portable FORTRAN programs and is generally compatible with Historian and Cray UPDATE, although there are some differences. Use of CMP for MELCOR configuration management is described in Section 4, Tools, Techniques, and Methodologies.) The program libraries generated by CMP are composed of named DECKs and COMDECKs, each containing numbered lines for the various MELCOR subprograms and common blocks, respectively, and grouped according to MELCOR package. Modifications to the program libraries are made in the form of IDENTs, which thereafter become a part of the program libraries and give a complete history of changes to these libraries.

FORTRAN source code for a particular computer or operating environment is generated from the MELCOR program libraries by CMP, and source routines are thus secondary, derived configuration items. Likewise, object libraries and executables are generated from the FORTRAN source by appropriate compilers and linkers for each computer system, and constitute secondary configuration items. Documentation, however, is a primary configuration item, matching as closely as possible the actual code. MELCOR documents are maintained as MASS-11 files.

MELCOR is undergoing active development and maintenance, which will likely continue for several years with intermittent releases of new baselined versions. This plan is intended to apply to all changes/updates to the code and documentation. A set of procedures has been established for review and approval of such updates. Section 3.2 of this SCM Plan contains an overview of these procedures, which center around the MELCOR Defect Investigation Report (DIR) Form.

MELCOR development and maintenance is primarily the responsibility of the Thermal/Hydraulic Analysis Division 6418 at Sandia National Laboratories. The MELCOR librarian has been designated as the principal person responsible for coordinating the maintenance and updating of the various MELCOR packages under the supervision of the MELCOR code development leader. When necessary, other members of the code development group may perform tasks normally carried out by the librarian, at the discretion of the code development leader. Once changes are approved, a set of procedural steps for actual implementation of the updates into the program libraries, documentation, FORTRAN source, object libraries, and executables is followed; detailed descriptions of these steps form the bulk of this plan.

This plan does not address the maintenance of the subsidiary utility procedures and software products (e.g., CMP) used to maintain and update MELCOR.

1.3 Definitions and Acronyms

CMP	Code Maintenance Package, the principal software tool used to generate and modify MELCOR program libraries.
DIR	Defect Investigation Report, the principal QA tool (termed in other places as a Software Change Request) for initiating, diagnosing, planning, reviewing, and implementing changes to MELCOR.

IFS	Integrated Files Store
Package	A group of MELCOR routines modeling a set of like phenomena or performing a well-defined set of functions.

1.4 References

Sandia Software Guideline, Volume 4, "Software Configuration Management"

ANSI/IEEE Std 828-1983

Quality Assurance Program Plan, Reactor Systems Safety Department 6410

"Machine-Dependent Coding"

MELCOR NUREG/CR-5531

"MELCOR Primer"

"MELCOR Software Configuration Management Procedures"

MELCOR Users' Guides

MELCOR Reference Manuals

MELCOR Architecture Programmers' Guide

2. MANAGEMENT

2.1 Organization

The amount of manpower devoted to development and maintenance of MELCOR is currently running between three and four Full-Time Equivalents (FTEs), spread over six to eight staff members. The MELCOR code development staff is currently encompassed by a single division at Sandia, Thermal/Hydraulic Analysis Division 6418. Under the oversight of the division supervisor, technical management of the project falls to the MELCOR code development leader. Several additional staff members work full or part time on code maintenance, user support, and new model development and implementation. Each staff member has primary responsibility for one or more MELCOR packages. One of these staff members acts as the principal liaison with the MELCOR user community, and resolves as many of the code problems encountered by the user community as is practical. Another staff member acts as the official MELCOR librarian administering the incorporation of changes to the code.

2.2 SCM Responsibilities

Code change requests may originate from the code development staff or the user community and are submitted on DIR forms through the process described in Section 3.2, Configuration Control. These are evaluated and prioritized by the code developer responsible for the MELCOR package encompassing the change, under the direction of the code development leader.

Configuration control of proposed changes to the code is accomplished within the division (the whole acting as a configuration control board, or CCB) through formal review of the updates by a code developer other than the one originating the updates, and approval by the code development leader or division supervisor (or alternate designated by the division supervisor).

The MELCOR librarian performs the bulk of the tasks directly related to configuration management, not only administering the process and designating the update identifier but also actually incorporating the updates into the controlled master program libraries for the various MELCOR packages and generating new FORTRAN source, object libraries, and executables, as specified within the MELCOR SCM Procedures (see Appendix A). The librarian periodically provides a status accounting for pending DIRs and processed updates, and is also responsible for maintaining the supporting utility procedures and software for updating MELCOR (i.e., CMP and utilities developed to utilize CMP).

2.3 Interface Control

MELCOR is currently a self-contained project, and no organizational or hardware interfaces exist. The only software interfaces that exist are the interface between the MELCOR plotting program MELPLT and the commercial graphics package DISSPLA, and the interfaces with various operating system-dependent routines called by MELCOR for functions such as date/time information and system error handling. The latter interfaces are dealt with in detail in the document "MELCOR Machine Dependent Coding." When code modifications are needed to interface MELCOR with new operating systems (e.g., UNICOS) or to coordinate with changes to old operating systems (e.g., changes to VAX/VMS), they are addressed when identified through the configuration control procedures described in Section 3.2.

2.4 SCM Plan Implementation

This plan was informally implemented with release of MELCOR 1.6.0 as a starting baseline. (Prior versions had been released, but changes were not managed through any formal configuration identification or control scheme.) Subsequent updates to MELCOR 1.6.0 have followed the basic configuration identification and control scheme described in this plan, although the plan was not formally published until after release of MELCOR 1.8.0. Periodically, the program libraries are resequenced at the release of a new baseline; that is, the change history included within the libraries is removed and they are distributed as fresh libraries with all inactive lines of code deleted and all active lines of code renumbered. (The nonresequenced libraries containing the change history are then archived on the Sandia Restricted Access Network (RAN) Common File System (CFS). The current baseline referenced by updates is MELCOR 1.8.0.

2.5 Applicable Policies, Directives, and Procedures

Other than the contents of this plan (with its companion SCM Procedures document), there are no other existing documents describing policies and procedures involving MELCOR SCM. This plan occasionally refers to other MELCOR documentation for supporting details.

3. SCM ACTIVITIES

3.1 Configuration Identification

A major release version of MELCOR is identified by a terminal zero in a 5-character name of the form 1.7.0, 1.8.0, etc. Each succeeding major release version incorporates a substantial increase in modeling capability or functionality from the previous version. Minor release versions are identified by a terminal digit other than zero, *e.g.*, 1.7.1, and incorporate new capabilities that are not quite as significant as those for major release versions. Both major and minor release versions are official release baselines that are distributed to users outside Sandia and which have been fully tested on a suite of release verification problems. All MELCOR packages in a release version of MELCOR are identified by the release identifier.

For ease of development and maintenance in a multiperson environment, MELCOR is divided into some twenty-odd packages. Each package is identified by a two-, three-, or four-character acronym or mnemonic identifier, generically referred to in this plan as "pkg" and listed in the MELCOR primer document. The primary MELCOR configuration items, the package program libraries, are each further subdivided into DECKs (Cray UPDATE and CMP parlance), one for each subprogram in the package, and into COMDECKs, for sections of code (typically common blocks) that occur identically in several places in MELCOR. Separate program libraries, one for DECKs (routines) and one for COMDECKs (common blocks), are maintained for each MELCOR package. DECKs are labeled the same as the subprogram name; COMDECKs are usually labeled the same as the common blocks they contain. Lines of code within DECKs and COMDECKs are identified by the DECK or COMDECK name and a line sequence number; this numbering information is contained within the program libraries generated by CMP.

Updates to a MELCOR package are assigned sequential alphabetic identifiers, starting with AA and generically referred to in this plan as "id", which are appended to the first three digits of the major release number "rel" (e.g., 1.8AQ where "rel" is 1.8 and "id" is AQ).

Update sets are made up of one or more IDENTs (CMP parlance again) used by CMP to modify a program library. Each IDENT refers to only a single DECK or COMDECK; the IDENT name is formed by appending the update identifier to the DECK or COMDECK name. Lines of code inserted by an IDENT are also identified by the IDENT name and a line sequence number. Code inserted and deleted by an IDENT becomes a part of the change history of the program library. DECKs and COMDECKs may also be purged from a program library by an IDENT. CMP program libraries are generated in ASCII text format so that they may be examined as needed using a text editor to obtain the detailed change history.

Typically, updates are implemented in small increments involving only one MELCOR package at a time. Updates may involve either new models and capabilities, or error correction of existing code. Internal versions of the constituent MELCOR packages for release version 1.8.0 are thus each identified by 5-character names of the form 1.8id. They are internal to Sandia and have had only limited testing, but the changes they embody have been promoted to official approval status (through step E on the DIR form) pending the next release of the code. These versions are created from an earlier internal version of the package by an update set 1.8id. This update set may also reference previous update sets, even so far as to totally undo the changes in a previous update. At the time of a new release and generation of a new baseline, all internal update sets may be replaced by a single update set identified by the new release number (e.g., "19") which has the net effect of all the changes made by the internal update sets.

The bulk of MELCOR documentation consists of three documents for each package: a Users' Guide, a Reference Manual, and a Programmers' Guide. A few additional documents cover special topics (e.g., this SCM Plan, or the MELCOR Primer). MELCOR documents are maintained in MASS-11 (a word processor) format and are identified by their title, the MELCOR version identifier to which they refer, and a date. Modifications to documents made in resolving a DIR are summarized in an appendix; at some later time, change bars may also be used to identify changes in a document from the previous released version.

SCM for MELCOR is accomplished on a VAX 8700 computer running the VAX/VMS operating system. Filenames correspond as closely as possible to the configuration identification scheme outlined above. Table 1 gives the filenames for the various MELCOR configuration items.

3.2 Configuration Control

A DIR is used to initiate and track the disposition of proposed changes to MELCOR. All types of changes, including correction of coding errors, revision or upgrade of physical models or numerical solution algorithms, addition of new models or features, and correction or revision of documentation, are resolved through the DIR procedures described in this section. A DIR either explicitly identifies or indicates the possibility of defects in coding, documentation, or input files. Thus, the term "defect" in the acronym "DIR" may refer to any actual or perceived shortcoming or imperfection, ranging from confusing documentation to major errors or missing models.

A DIR may evolve through as many as five steps, which are designated by the letters A through E

A. Request/Description
B. Diagnosis
C. Resolution Plan
D. Changes/Testing
E. Update Implementation

Table 1. MELCOR CI Filenames

pkg_RTN.SRC	CMP source file for "pkg" routines (contains DECKs with common blocks replaced by *CALL directives)
pkg_CBK.SRC	CMP source file for "pkg" common blocks (contains COMDECKs)
pkg_RTN_relid.PRL	CMP program library for "pkg" version "relid" routines
pkg_CBK_relid.PRL	CMP program library for "pkg" version "relid" common blocks
pkg_RTN_id.UPD	CMP update set "id" (contains *INSERT, *DELETE, and *PURGE directives) to modify "pkg" routines in pkg_RTN_relid.PRL
pkg_CBK_id.UPD	CMP update set "id" (contains *INSERT, *DELETE, and *PURGE directives) to modify "pkg" common blocks in pkg_CBK_relid.PRL
pkg_RTN_id.XAD	CMP update set "id" (contains new DECKs) to add "pkg" routines to pkg_RTN_relid.PRL
pkg_CBK_id.XAD	CMP update set "id" (contains new *COMDECK directives) to add "pkg" common blocks to pkg_CBK_relid.PRL
pkg_relid.FOR	FORTRAN source file (CMP compile file) for "pkg" version "relid" generated from pkg_RTN_relid.PRL
pkg_relid.CBK	FORTRAN common blocks for "pkg" version "relid" generated from pkg_CBK_relid.PRL
pkgL.OLB	Object library for version "relid" generated from optimized compilation of pkg_relid.FOR
pkgLD.OLB	Object library for Version "relid" generated from non-optimized debug compilation of pkg_relid.FOR

Note: Periods in "relid" are replaced by dashes ("-") in filenames.

A complex DIR involving much time at steps B and/or C will require review and approval for each step before proceeding to the next. On the other hand, for a simple DIR (i.e., one involving little time to diagnose the defect and make the necessary modifications), it is permitted to proceed all the way through step D before submitting the DIR for review and approval, at the discretion of the code development leader.

The person responsible for preparing a DIR for approval at each step is referred to as the investigator and is assigned by the code development leader. More than one investigator may be involved in a given step, and different investigators may be responsible for successive steps of a DIR. Review of a DIR step must be carried out by one or more code developers other than the investigator(s), who are assigned by the code development leader. Approval of a step is normally given by the code development leader, unless he is the investigator of that step. In that case, approval must be given by the division supervisor or an alternate designated by him.

DIRs may include both typed and legibly handwritten information, as well as plots and other information needed to fully describe the defect, its diagnosis, and the plan for resolving it. Differences between the modified and original coding, generated by automatic utilities such as the DCL DIFFERENCES command, are attached to the DIR form. When a CMP update set is generated by the MELCOR librarian after approval of one or more DIRs, it is also attached to the DIRs before filing.

In step A, a request for resolution of a defect is recorded; the requester may be anyone using or reviewing the code or documentation. The requester should complete the following information on the DIR form for step A: TITLE (brief description); VERSION (of code or documentation); and REQUESTER, REQUEST, and REQUEST BASIS. The form is then given to the MELCOR librarian, who logs the request, assigns it a number, checks it for completeness, and makes a copy for the master DIR file. The code development leader then assigns an investigator for step B, who is given the original DIR.

The request should describe the difficulty encountered by the requester, any new features or other changes being requested, and the reasons for such changes. If the request is based on results obtained during code execution, input and output files required to identify and diagnose the defects must be accessible to the SNL Code Development Group in order to proceed with step B. Outside requesters should supply such files via IBM-compatible floppy disks. Any related DIRs known by the requester should be noted; the code development leader may note additional related DIRs. If more information is needed before investigation of the defect commences, the code development leader will request it, and if the investigator needs additional material after diagnosis begins, he may request it.

The investigator assigned for step B diagnoses the source of the problem and describes it fully. Any relevant files should be transferred from floppy disks to SNL computer storage; the investigator should record these file names in the DIR for later testing and reference by those assigned to review the DIR. In many cases, step B will be trivial; however, care must be exercised to determine the root cause of the defect. In a number of situations, the diagnosis may be complicated by the interaction between MELCOR packages, and this may in fact be the most difficult and time-consuming step in resolving the DIR. Due to this difficulty, there may be more than one investigator for step B. For example, the first investigator may track down the general problem to a specific package. The package developer may then diagnose the problem further to locate the defect. During the diagnosis, other defects may also be discovered. These should be noted here and addressed in step C.

In step C, a plan to resolve the defect is formulated and described. For simple changes (i.e., typos, etc.) this step will be trivial. However, for some complex problems, the plan for resolution is important; there may be a variety of alternatives to resolve a defect, one of which must be chosen

and justified. Additional defects identified during diagnosis may be resolved with this plan; if they are not, reference should be made to specific DIRs submitted describing the defects for later resolution. Also, if the defect is only partially resolved, a revised DIR should be submitted for later full resolution of the problem. If a defect has already been resolved by a previous update, reference to that update identifier should be made. Plans for resolving DIRs that involve more than a modest amount of code or documentation changes must be reviewed by another code developer and approved by the code development leader before commencing with step D.

The actual coding and/or documentation changes necessary to resolve the defect are made and documented in step D. Changes to the code must be specified at a level of detail that is at least sufficient for the DIR reviewer to fully understand their impact. This typically means attaching a set of differences, automatically generated by the DCL DIFFERENCES command, with appropriate highlighting and other notations generated manually to assist the reviewer in following the changes. New or replacement routines must be documented sufficiently in step C and internally via COMMENT lines for the reviewer to fully understand them as well. Changes to documentation should include complete paragraphs or sections of modified and original text, with differences highlighted in some fashion.

Testing of code changes is also conducted during step D and should be explicitly noted here. This testing should verify that the changes resolve the defect as described in step C and (as far as practical) that they do not introduce new errors into the code. The investigator may also identify additional, more extensive testing that could be performed after approval of the changes to fully validate a new code model. Separate effects tests of the package alone should be identified separately from integral tests involving the entire code.

In step E, the CMP update set is prepared by the MELCOR librarian from the modified versions of the source code generated by the code developer(s) during step D. New versions of the program libraries, FORTRAN source code, object libraries, and executable code are then generated. Specific procedures have been established to ensure that the changes actually made by the code developer(s) are accurately reflected in the program libraries. An internal document, "MELCOR Software Configuration Management Procedures," describes these procedures in detail. The procedural details change frequently, and so are not included as a part of the main body of this SCM Plan. However, the version of these procedures current at the time of publication of this SCM Plan is included as Appendix A.

3.3 Configuration Status Accounting

The MELCOR librarian is responsible for generating and distributing information on the status of DIRs and updates to the code on a periodic basis. To accomplish this, three log files are maintained. The DIR log lists the DIRs in order by the numbers assigned to them, and includes the DIR's title, the date it was submitted, the date it was approved (or shows the investigator responsible for a pending DIR), and the update identifier implementing the changes. The update log lists each update in order by its identifier, and includes a list of all DIRs (number and title) resolved with the update and the date the update was approved. The package log lists each MELCOR package along with all identifiers for updates referencing that package.
After an update is approved, the investigator for step D in the DIR procedures must generate a brief summary (no more than a few sentences) of the changes effected by the update and their impact on existing and future MELCOR calculations. This information is reported to users by the MELCOR

librarian after implementation (step E) and is also used in the appropriate status and progress reports distributed periodically to users.

At the beginning of each month, a current status report that summarizes changes in the above log files is generated and distributed to the MELCOR code development group. This report contains a master list of all pending DIRs (number, title, date submitted, and investigator), a separate list grouping the pending DIRs by package, and a list of updates/DIRs processed for the previous month. The latter list should also contain the information describing the changes affected by the updates and is also included in the MELCOR project progress reports.

3.4 Audits and Reviews

An internal physical configuration audit will be performed immediately prior to any new release of MELCOR to external users. This audit will consist of each code developer reviewing the program libraries of the packages assigned to him to verify that each subroutine and common block modified or added by him is represented by an appropriate IDENT or DECK in the program library. To accomplish this, all changed versions of routines and common blocks, along with their corresponding original versions, must be retained by the code developers until this audit is performed.

A functional configuration audit will also be performed prior to release of a new version of MELCOR. For a minor version release, this will consist of verification that the code satisfactorily executes the official MELCOR sample problem. For a major version release, this will consist of verification that the code satisfactorily executes a more extensive set of validation and verification test problems. These test problems will be chosen beforehand by the MELCOR code development leader and division supervisor, and will include the set of problems successfully executed by MELCOR for the previous release.

4. TOOLS, TECHNIQUES, AND METHODOLOGIES

The cornerstone tool used for the configuration management of MELCOR is the CMP (Code Maintenance Package) set of programs, which provides a convenient and portable method of maintaining computer programs and other data sets. CMP provides an update processor, update correction set generation by file comparison, deck sorting, and various file handling utilities. Library files can be written in a system independent mode and transferred between different computer systems without loss of correction history. CMP is written in strict FORTRAN 77 and can be used on any system with sufficient memory and a full standard compiler.

CMP is used to generate and modify the controlled ("project") libraries for the MELCOR packages. Upon release of a version of MELCOR, these libraries are archived as static libraries on the Sandia Integrated File Storage (IFS) system and on copies of the release package stored on magnetic tape. When making test modifications to MELCOR, a code developer must access the project libraries to obtain the latest official versions of the routines or common blocks to be modified; the collection of these files for the typical variety of modifications in process constitute the developer's dynamic (or programmer's) library.

A set of DCL command procedures has been written to substantially automate the process of using CMP to maintain MELCOR. The GENERATE_UPDATES command procedure is used to generate update correction sets. Updating the program libraries is accomplished by the UPDATE command procedure, and generation of FORTRAN source for a given computer and operating system is done through the GENERATE_FORTRAN command procedure. These command procedures are built around the conventions established for the naming of MELCOR packages, versions, update identifiers, file names, etc., and they simplify the process a great deal.

The GENERATE_FORTRAN command procedure also uses a separate utility program developed for MELCOR called MELUTL. Separate program libraries are maintained for the common blocks for each package. Applying GENERATE_FORTRAN to these common block libraries produces a set of files containing the current versions of these common blocks. The function of MELUTL is to then substitute these current versions of the common blocks into the FORTRAN source being generated during application of GENERATE_FORTRAN to the program library for a package's routines.

A companion document to this Plan, "MELCOR Software Configuration Management Procedures," is located in Appendix A. It lists the procedural steps for a code developer and the MELCOR librarian to follow when modifying a MELCOR package and implementing the corresponding updates into the project libraries. The MELCOR SCM Procedures document is updated frequently as the MELCOR project continues to automate its procedures for maintaining the code. Therefore, these procedures have not been made a part of the body of this plan, but the version of these procedures current with publication of this plan is given in Appendix A.

5. SUPPLIER CONTROL

All code written by third-party suppliers, such as other national laboratories, universities, and subcontractors, must go through the DIR procedures described in Section 3.2, Configuration Control. Upon review and approval, no distinction is made between this code and that generated within the MELCOR code development group.

6. RECORDS COLLECTION AND RETENTION

All DIR and update paperwork, electronic files containing MELCOR baseline program libraries, update sets, and log files, and copies of official release packages distributed to users via magnetic media are to be retained for the lifetime of the MELCOR code. The paperwork and release packages are to be stored in clearly labeled boxes and kept in a safe but convenient place (typically the code development leader's office). Electronic files are to be archived on the Sandia IFS system.

DIR

MELCOR Defect Investigation Report

Title: _____	*DIR Number:* _____
	Date sub. : _____
Requestor/Org.: _____	*Update ID :* _____
Code Version : _____	*Date appr.:* _____

Request basis	*Changes required*	*Severity*
☐ Error ☐ Revision ☐ New feature ☐ Other	☐ None ☐ Documentation ☐ Coding ☐ Input deck	☐ Minor ☐ Major ☐ Medium

Step	Investigated	Date	Reviewed	Date	Approved	Date
A. Request						
B. Diagnosis						
C. Plan						
D. Changes/Testing						
E. Implementation						

Appendix A to MELCOR SCM Plan

MELCOR Software Configuration Management Procedures

February 16, 1990

Randall M. Summers
Thermal/Hydraulic Analysis
Division 6418
Sandia National Laboratories
Albuquerque, NM 87185

This document lists the procedural steps for a MELCOR code developer and the MELCOR librarian to follow when modifying a MELCOR package and implementing the corresponding updates into the controlled ("project") libraries. These procedures constitute a companion document to the MELCOR Software Configuration Management Plan; that document discusses the more general concepts important to MELCOR SCM. MELCOR is developed and maintained principally on a VAX 8700 computer running the VAX/VMS operating system; these procedures for MELCOR SCM are determined in large part by this computing environment. Approximately 200,000 512-byte VAX blocks are required for SCM of MELCOR. DCL command procedures maintained by the MELCOR librarian (UPDATE, GENERATE_UPDATES, and GENERATE_FORTRAN) use the Sandia-developed Code Maintenance Package (CMP) for the automatic generation of program libraries, updates, and Fortran source, respectively.

Configuration changes to MELCOR documentation are handled separately, under a different set of procedural steps. These steps have not as yet been specifically determined. Note, however, that review and approval of documentation changes are still handled within the DIR review process.

In the following steps, lower case strings appearing in filenames and so forth are meant to indicate replacement of the string by a specific package name, update identifier, etc; upper case strings should be used verbatim. Example VAX/VMS commands are given in each of these steps for updating several COR package routines and common blocks. These examples are illustrative only and do not constitute the SCM procedures, and should not be slavishly followed. In these examples, the assigned update identifier "id" is BG (the long form, including the baseline version to which the update refers, would be 1.8BG); the last previous identifier is 1.8AQ ("oldrtnid") for the COR package routines and ("oldcbkid") 1.8AZ for the COR package common blocks. It is recommended that the procedures UPDATE, GENERATE_UPDATES, and GENERATE_FORTRAN be exercised in interactive prompt mode by issuing only the UPD, GUPD, and GFOR commands without parameters, rather than using parameters on the command line as shown in this document.

Procedural Steps for the Code Developer

1. Create a new subdirectory of the form [...DIRnum] to process the DIR(s). Only files related to the DIR(s) being processed should be stored in this subdirectory. Also, since the GENERATE_UPDATES utility procedure used by the MELCOR librarian will work properly on only one package at a time, it should be clear which files belong to each package. (This is generally the case, since most routines in most packages begin with the package identifier "pkg"; three notable exceptions are the Cavity (CAV), Executive (EXEC), and Utility (UTIL) packages.) If identification of files with their respective packages is not clear, separate subdirectories should be used for those packages.

```
$ CREATE/DIR [RMSUMME.MELCOR.V180.DIR612]
$ SET DEF [RMSUMME.MELCOR.V180.DIR612]
```

2. Create individual Fortran files from the old version program libraries for each package routine (DECK) to be modified; these files should be named deckname.ORG, where "deckname" is the name of the DECK (and hence the routine). Also, create a file as well for the old version common blocks if any need to be modified; this file should be named pkg_CBK.ORG. Creation of these files is best accomplished by applying the GENERATE_FORTRAN utility procedure to the latest CMP program libraries (pkg_RTN_oldrtnid.PRL and pkg_CBK_oldcbkid.PRL files).

   A code developer should make sure that he himself does not have updates pending or changes in process that affect routines or common blocks about to be modified. Furthermore, any code developer making changes to a package for which he does not have primary responsibility must verify with both the MELCOR librarian and the code developer who does have primary responsibility that there are no other updates pending or changes in process that affect these routines or common blocks. The librarian will act as a central clearinghouse for all pending updates.

   If other updates affecting these routines or common blocks are pending implementation, or some other developer is in the process of modifying them, other means for generating the .ORG files may be used, as appropriate to the situation. However, much care should be exercised to get the proper routines or common blocks and to avoid introducing extraneous lines or making other inadvertent changes. In particular, extraneous blank lines must not be inserted, as may happen when writing a Paste buffer using SLEM or EDT. Generation of the correct .ORG file will be double-checked by the MELCOR librarian after DIR approval when updates are generated for actual implementation into the program libraries.

```
$ GFOR COR CBK 1.8AZ LIST VAX LIBS DIRCOR:
$ RENAME COR_1-8AZ.CBK COR_CBK.ORG
$ GFOR COR RTN 1.8AQ LIST VAX LIBS DIRCOR:
 CORRN1
   .
   .
$ (split COR_1-8AQ.RTN into CORRN1.ORG, etc.)
      (no automated procedure developed to split these out yet)
```

3. Make the desired modifications to routines and common blocks using a text editor. These modifications should follow the plan outlined in step C of the Defect Investigation Report (DIR) form and must receive sufficient testing to verify that the defect is adequately resolved. Modified routine files should be named deckname.FOR, and the modified common block file should be named pkg_CBK.MOD. Replacement routines should be named deckname.RPL, new routines should be named deckname.NEW, and deleted routines should be named deckname.DEL. Table 1 summarizes the files created by the code developer.

Table 1. Filenames Created by Code Developer

deckname.FOR	Modified Fortran source routines
deckname.NEW	New Fortran source routines
deckname.RPL	Replacement Fortran source routines
deckname.DEL	Deleted Fortran source routines (this may be an empty file)
deckname.ORG	Most recent Fortran source routines for modified, replaced, and deleted decks
deckname.DIF	File containing differences between the .ORG files and the .FOR or .RPL files
pkg_CBK.MOD	Replacement file for common blocks
pkg_CBK.ORG	Most recent common block file
pkg_CBK.DIF	File containing differences between the pkg_CBK.ORG and pkg_CBK.MOD files

The code developer must explicitly identify in step D of the DIR form all routines and packages that have not been changed directly but which are affected by changes in common blocks. New Fortran source must be generated for these routines and packages by the MELCOR librarian, even though the program libraries have not been changed for these DECKs.

The code developer also must explicitly state in step D of the DIR form whether any modifications to code within *IF DEF blocks have been made. These changes must receive special attention, perhaps generating the updates by hand, since the GENERATE_UPDATES utility procedure used by the MELCOR librarian may not adequately handle them.

New common blocks inserted in pkg_CBK.MOD must be placed in alphanumeric COMDECK order to facilitate checking by the MELCOR librarian.

```
$ EDT/OUT=.MOD COR_CBK.ORG
$ EDT/OUT=.FOR CORRN1.ORG
        .
        .
```

4. Generate text differences for the modified routines and common blocks using the VAX/VMS DIFFERENCES command; the files should be named deckname.DIF and pkg_CBK.DIF, and should be appended to the DIR form.

```
$ DIF/OUT=.DIF CORRN1.FOR .ORG
        .
        .
$ DIF/OUT=.DIF COR_CBK.MOD .ORG
```

5. Submit to the code development leader a DIR Form filled out through step D with .DIF files and any other pertinent information (e.g., plots comparing runs with old and new routines) attached for checking and approval. If the effects of the modifications are not clear from the DIFFERENCES output only, full subroutines should also be included.

```
$ IMP/TWO/LAND *.DIF
```

6. Upon DIR approval and assignment of a two-letter update identifier ("id") by the MELCOR librarian, write a short summary description of the update in ASCII text and store in file UD3:[MELCOR.LOG]UPDid.TXT. Include any pertinent information regarding the expected effect on new or existing calculations, especially with regard to incompatible restart files. When the update is implemented by the MELCOR librarian, this file will be mailed to MELCOR users and included in condensed form in progress reports and the users newsletter.

```
$ EDT UD3:[MELCOR.LOG]UPDBG.TXT
```

7. Verify that the all updates to resolve the DIR have been completely and properly implemented into the controlled program libraries and initial under the "Reviewed" block under step E on the DIR Form. Completing this step will involve viewing the program libraries, which are created as ASCII text files. **DO NOT DELETE .FOR, .RPL, .NEW, AND .ORG FILES.**

Procedural Steps for the MELCOR Librarian

1. A two-letter update identifier "id" (sequential to the previous identifier assigned for all of MELCOR, not just the package being modified) must be assigned. Create one or more new subdirectories to process the update, with subdirectory names of the form [MELCOR.pkg.V180.id]. Only files related to the DIR(s) being processed should be stored in these subdirectories. Since the GENERATE_UPDATES utility procedure used below will work properly on only one package at a time, each package to be modified must have a separate subdirectory.

```
$ CREATE/DIR [MELCOR.COR.V180.BG]
$ SET DEF [MELCOR.COR.V180.BG]
```

2. Copy the .FOR, .RPL, .NEW, .DEL, and .MOD files from the subdirectory referenced in the DIR form for each package. Also, create temporary .CHK files for each routine by applying the GENERATE_FORTRAN utility procedure to the latest CMP program libraries (pkg_RTN_oldrtnid.PRL and pkg_CBK_oldcbkid.PRL files) and check for differences with the .ORG files used by the code developer as his starting point. There must be zero differences when trailing blanks are ignored. Once this has been verified, the .CHK files may be deleted, but the .DIF file showing the check was made must be retained for auditing purposes. Extraneous blank lines may be eliminated if necessary by the MELCOR librarian, but should not exist if the code developer properly followed step 2 of his procedures.

```
$ COPY
[RMSUMME.MELCOR.V180.DIR612]*.FOR,*.RPL,*.NEW,*.DEL,*.MOD

$ GFOR COR CBK 1.8AZ LIST VAX LIBS DIRCOR:
$ RENAME COR_1-8AZ.CBK COR_CBK.CHK
$ DIF/IG=TR/OUT COR_CBK.CHK
[RMSUMME.MELCOR.V180.DIR612]COR_CBK.ORG
$ TYPE COR_CBK.DIF

$ GFOR COR RTN 1.8AQ LIST VAX LIBS DIRCOR:
 CORRN1
   .
   .
$ RENAME COR_1-8AZ.RTN COR_RTN.CHK
$ COPY [RMSUMME.MELCOR.V180.DIR612]*.ORG/EXCL=COR_CBK
COR_RTN.ORG
$ DIF/IG=TR/OUT COR_RTN.CHK .ORG
$ TYPE COR_RTN.DIF

$ DELETE *.CHK;*,*.ORG;*
```

3. Generate CMP updates for the modified routines (pkg_RTN_id.UPD and pkg_RTN_id.XAD files) and common blocks (pkg_CBK_id.UPD and pkg_CBK_id.XAD files) using the GENERATE_UPDATES utility procedure. Special attention must be given to any modifications of code within *IF DEF blocks, perhaps even creating the updates by hand, since GENERATE_UPDATES may not adequately handle them. Modifications to code within *IF DEF blocks must have been indicated by the code developer on the DIR form under step D.

   The CMP updates that are generated should be quickly compared with the differences output supplied by the code developer to verify that nothing is amiss and that faulty matches have not been made, thus creating many changes unnecessarily. If that happens, CMPDIF may be executed again with a different matching parameter to reduce the size of the IDENT.

   *PURGEs generated by processing of the .RPL files must be manually replaced by the command *DELETE firstline,lastline. Also, the corresponding *DECK and *COMDECK directives in the .XAD files must be changed to *IDENT decknameid, with appropriate *INSERT directives, and moved, along with the new routine and common block lines, into the .UPD files. This will retain the history of the deleted routine or common block for comparison with its replacement.

   ```
   $ GUPD COR CBK 1.8AQ BG DIRCOR:
   $ GUPD COR RTN 1.8AZ BG DIRCOR:
   ```

4. Copy the .UPD and .XAD files for each package to its LATEST subdirectory (defined by logical symbol "DIRpkg") and go to that subdirectory to update the program libraries, Fortran source, and object libraries.

   ```
   $ COPY *.UPD,*.XAD DIRCOR
   $ SET DEF DIRCOR
   ```

5. Generate new CMP program libraries for each package's routines and common blocks (pkg_RTN_updid.PRL and pkg_CBK_updid.PRL) using the UPDATE utility procedure. The string "updid" is the long form that includes the baseline version number to which the update refers (e.g., 1.8). The previous program libraries should be saved on the IFS for possible error recovery and then deleted from the VAX to save space. Some libraries stored on the IFS may be deleted periodically if gaps between versions that are retained would not cause a major inconvenience in recreating a specific version. Retaining every third or fourth version of a program library on the IFS, in addition to the second most recent, is suggested as being reasonable.

   ```
   $ UPD COR CBK 1.8AZ BG DIRCOR:
   $ UPD COR RTN 1.8AQ BG DIRCOR:
   ```

6. If common blocks for a package have been changed, generate a new common block file for VAX (pkgLATESTVAX.CBK) from the newly created program library. This must be

created prior to generating any of the routines to ensure that the latest common blocks are incorporated into the routines. Compare the new common blocks with the modified common blocks (.MOD file) created by the code developer. Except for non-VAX/VMS code (bracketed by *IF DEF directives) that is inserted by the update, there must be zero differences when trailing blanks are ignored. This is a double check to ensure that the utility procedures have functioned properly with no errors. The DIFFERENCES output file (.CBKDIF) showing the check was made must be retained for auditing purposes.

```
$ GFOR COR CBK 1.8BG LIST VAX LIBS DIRCOR:
$ DIF/IG=TR/OUT=.CBKDIF COR_1-8BG.CBK [-.BG]COR_CBK.MOD
$ TYPE COR_1-8BG.CBKDIF
$ COPY COR_1-8BG.CBK CORLATESTVAX.CBK
```

7. As a double check to ensure that the utility procedures have functioned properly with no errors, generate VAX Fortran for routines that have been modified (pkg_updid.FOR) from the newly created program libraries. Note that the GENERATE_FORTRAN utility gives the .RTN extension by default; this should be renamed to .FOR for VAX Fortran. Compare these routines with the modified routines (.FOR, .RPL, and .NEW files) created by the code developer. Except for non-VAX/VMS code (bracketed by *IF DEF directives) that is inserted by the update, there must be zero differences when trailing blanks are ignored. The DIFFERENCES output file (.RTNDIF) showing the check was made must be retained for auditing purposes.

```
$ GFOR COR RTN 1.8BG LIST VAX LIBS DIRCOR:
 CORRN1
   .
   .
$ RENAME COR_1-8BG.RTN .FOR

$ COPY [-.BG]*.*/EXCL=*.*D* COR_1-8BG.CHK
$ DIF/IG=TR/OUT=.RTNDIF COR_1-8BG.FOR .CHK
$ TYPE COR_1-8BG.RTNDIF
$ DELETE COR_1-8BG.CHK;*
```

8. Generate new Fortran common blocks for the CRAY (pkgLATESTCRAY.CBK) from the newly created program libraries.

```
$ GFOR COR CBK 1.8BG LIST CRAY LIBS DIRCOR:
$ RENAME COR_1-8BG.CBK CORLATESTCRAY.CBK
```

9. Generate new Fortran routines for both VAX and CRAY (pkg_updid.FOR and pkg_updid.RTN, respectively) from the newly created program libraries. This step is distinguished from Step 7 in that all routines involving any common blocks that have been changed are included, not just routines. The MELCOR librarian may also choose during this step to generate Fortran source for the entire package using the ALL qualifier instead of

LIST with the GENERATE_FORTRAN utility. Note again that this utility gives the .RTN extension by default; renaming to .FOR should be done only for VAX Fortran, so **be careful**. (If no additional routines are necessary or desired for compilation and generation of new object libraries, duplicating the VAX Fortran created in Step 7 is not necessary.)

```
$ GFOR COR RTN 1.8BG ALL VAX LIBS DIRCOR:
$ RENAME COR_1-8BG.RTN .FOR
$ PURGE COR_1-8BG.FOR

$ GFOR COR RTN 1.8BG LIST CRAY LIBS DIRCOR:
 CORRN1
```

10. Compile VAX routines twice: once with /OPT qualifier and once with /NOOPT/DEBUG/CHECK=(BOUNDS,OVERFLOW), both times with STANDARD=ALL. Warnings related to VAX-specific code (e.g., READONLY in OPEN statements) may be ignored; other warnings or errors must be resolved.

```
$ FOR/NOLIST/STAND=ALL/OPT/OBJ=.OPT COR_1-8BG
$ FOR/NOLIST/STAND=ALL/NOOPT/DEBUG/CHECK=(BOUNDS,OVERF)
COR_1-8BG
```

11. Update VAX optimized and debug object libraries (pkgL.OLB and pkgLD.OLB files) with newly created object files. For nontrivial changes, and especially for changes modifying the data base, save the old object libraries for a period of one week, renamed as pkgL_oldid.OLB and pkgLD_oldid.OLB. If Fortran source for the whole package has been generated, use the LIB/CREA command; otherwise use the LIB/REPL command. Once object files have been incorporated into an object library, they may be deleted.

```
$ COPY CORL.OLB CORL_1-8AQ.OLB
$ COPY CORLD.OLB CORLD_1-8AQ.OLB
$ LIB/REPL CORL COR_1-8BG.OPT
$ LIB/REPL CORLD COR_1-8BG.OBJ
$ DELETE COR_1-8BG.OPT;*,COR_1-8BG.OBJ;*
```

12. Create new optimized and debug MELGEN and MELCOR executables in UD3:[MELCOR.EXE] using the LINKGEN, LINKGEND, LINKCOR, and LINKCORD utility procedures. The .EXE filenames should include the version identifier.

```
$ SET DEF UD3:[MELCOR.EXE]
$ LINKGEN 1-8BG
$ LINKGEND 1-8BG
$ LINKCOR 1-8BG
$ LINKCORD 1-8BG
```

13. Store CRAY Fortran source created in Step 9 on IFS (in VAX native text, upper case) on node /MELCOR/V180/LATEST/pkg_updid.RTN. The CRAY Fortran source can then be deleted from the VAX.

```
$ NETON
$ MASS
? DEFAULT DIR=/MELCOR/V180/LATEST
? STORE COR_1-8BG.RTN
? END
$ NETOFF
$ DELETE COR_1-8BG.RTN;*
```

14. Create updated CRAY object (build) library and new MELCOR and MELGEN absolutes using the CCL procedures BOLB and BEXE (run them in interactive prompt mode). These absolutes should be stored on IFS (the procedures prompt for this).

```
/ upld bolb ud3:[melcor.cmp]bolb.com
/ upld bexe ud3:[melcor.cmp]bexe.com
/ bolb
/ bexe
```

15. Fill out and attach the official MELCOR Update cover sheet, along with listings of the updates that were generated, and file all of the DIR paperwork in sequential update order (not DIR order). Initial the "Investigated" block on Step E (Implementation) of the DIR Form, and return it to the code developer that processed the changes for verification that the updates were completely and properly implemented. Copies of revised or new documentation should be retained separately for distribution to users.

```
$ IMP/TWO/LAND COR_*_BG.UPD,COR_*_BG.XAD
```

16. MAIL the message generated by the code developer regarding the update (stored in [MELCOR.LOG]UPDid.TXT) to MELCOR users informing them of the implementation of the update into the latest libraries.

```
$ MAIL/SUBJ="MELCOR UPDATE 1.8BG" - [MELCOR.LOG]UPDBG.TXT
@MAIL.USR
```

17. Modify the entries in the DIR log files to reflect approval of the update resolving the DIRs, and add the update to the update log file.

```
$ EDT [MELCOR.LOG]DIRLOG.TXT
$ EDT [MELCOR.LOG]UPDLOG.TXT
$ EDT [MELCOR.LOG]PKGLOG.TXT
```

Appendix F
Software Configuration Management Plan Template

General System
Software Configuration Management Plan

Template for General Use

This template for a project Software Configuration Management Plan is structured after ANSI/IEEE Std 828-1990. Std 828-1990 is slightly different in organization than its predecessor, Std 828-1983. However, the core content of information required is unchanged.

The reader must tailor the presentation of information this template contains to meet the needs of the project at hand. While a standard format is both desirable and appropriate for Sandia, the presentation sequence and style is subject to the needs of the plan's users and the project requirements. The six sections of information should be included in any such restructured plan. Small projects may find it advantageous to consolidate the sections and make the Software Configuration Management Plan a chapter in an overall Software Quality Plan or Software Development Plan. Large complex projects may need to expand the subdivision of information presented to adequately address those complexities.

The template is available on electronic media and can be translated to almost any word processor format. Section headings are in bold type. Italics type is instructional information that should be deleted when a section is filled in with the necessary information for the project.

Italicized text within this document is meant to provide information only and should not be left in the document when it is completed. Standard text may remain in the document.

Contents

1. Introduction

1.1 Purpose

Briefly state why this plan exists and who the intended audience is.

1.2 Scope

Briefly describe the software development project. Define what this plan will and will not address. Identify the software products that will be addressed in this plan.

Software products include the Software Requirements Specification, Design Information, Source Code and Executables, Off-the-Shelf (OTS) or Vendor Software, Compilers, Editors, Computer-Aided Software Engineering (CASE) tools, Test Procedures and Test Data, Interface Documents.

Other products that can be controlled under this plan are the contractual documents that fostered the project. Identify the organizations involved and the time frame of the project.

For embedded software, indicate the relationship between software configuration management (SCM) and the hardware or system configuration management for the project.

1.3 Definitions and Acronyms

Reference ANSI/IEEE 610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology" as the source for common terms used in this plan. Define only new terms or terms used in a unique way in this plan, for example:

Interface Control. *The process of (1) identifying all functional and physical characteristics relevant to the interfacing of two or more CIs provided by one or more organizations, and (2) ensuring that proposed changes to these characteristics are evaluated and approved prior to implementation.*

Supplier. *Any project, organization, or other group (e.g., matrixed group, another division, department) external to the management organization(s) in this plan that provides software/firmware/hardware components necessary to the success of the software project(s) within the scope of this plan.*

Provide a list of acronyms so they can always be found in one location, for example:

CASE	*Computer -Aided Software Engineering*
CCB	*Configuration Control Board*
CI	*Configuration Item*
FCA	*Functional Configuration Audit*
IEEE	*Institute of Electrical and Electronic Engineers*
LAN	*Local Area Network*
OTS	*Off-the-Shelf*

PCA *Physical Configuration Audit*
PROM *Programmable Read Only Memory*
SCCB *Software Configuration Control Board*
SCM *Software Configuration Management*
UART *Universal Asynchronous Receiver/Transmitter*

1.4 References

List the documents cited elsewhere in this plan. Also list governing documents such as contractual standards and directives if external customers are involved. Referencing existing policies, practices and procedures directly applicable to this plan will avoid reiterating similar information. A distinction should be made between references necessary to execute this plan and general or supplementary information. Sample general references might include:

[1] Sandia National Laboratory Quality Plan.
[2] Sandia Software Guidelines, Vols 1, 3, 4 and 5
[3] SNL Software Development Policy and Recommended Software Development Practices, L. D. Bertholf memorandum, November 19, 1990
[4] ANSI/IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.
[5] ANSI/IEEE Std 828-1990, IEEE Standard for Software Configuration Management
[6] ANSI/IEEE Std 1042-1987, IEEE Guide to Software Configuration Management.

Some governing documents that may be applicable for DoD reimbursable programs include:

[7] DOD-STD-2167A, Military Standard, Defense System Software Development, Chapter 4.5, Software Configuration Management.
[8] DOD-STD-2168, Military Standard, Defense System Software Quality Program.

Some references that may be necessary if the Engineering Release System and (Sandia) Drawing System are used include

[9] EP401033, "Revising Drawings and Part Numbers to Define Product Changes."
[10] EP401035, "Sandia and Production Agency Acceptance Equipment Interfaces."
[11] EP401043, "Acceptance Equipment Program Definition."
[12] EP401044, "Engineering Release System."
[13] EP401045, "Definition of Computer Software Configuration Items."
[14] EP401054, "Nine-Digit Part Number System."

2. Management

2.1 Organization

Provide an organizational chart of the project structure if this is a stand-alone plan. Describe the functional roles of each managerial and technical organization , the relationships between organizations, and how SCM fits into the organizational structure .

2.2 SCM Responsibilities

Describe who (ie., what organizational entity or project position) is responsible for the activities in Section 3, ie., who has responsibility for the software component(s); who will provide representation to the Software Configuration Control Board (SCCB) and the system Configuration Control Board (CCB), if it exists. (If no system CCB exists, then the SCCB is usually referred to as just the CCB); who is responsible for identifying and controlling the software, who has the authority to release software, data, or documentation, who will perform audits and provide status accounting information as necessary, who is responsible for interface control within the system and external to the system; who is responsible for vender/supplier software.

2.3 Applicable Policies, Directives, and Procedures

Identify any external policies or procedures and describe how they affect this plan. This is the section to interpret how these factors influence or constrain the approach taken in this plan.

3. Software Configuration Management Activities

Note: The traditional SCM activities are Configuration Identification, Configuration Control, Configuration Status Accounting, and Audits and Reviews. The new Std 828-1990 has moved into this section "other" activities. Interface Control was moved from Section 2. Supplier (subcontractor, vendor, supplier) Control was moved from Section 5. Records Collection and Retention (Section 6 in the old Std 828-1983) was deleted but has been included in this section by the author.

3.1 Configuration Identification

Define what software will require configuration identification and at what level will the establishment of a Configuration Item (CI) begin. The size of the project has little bearing on the necessity to accurately establish configuration identification.

Describe how the process of configuration identification will be accomplished. The information in this section should document an identification scheme that reflects the structure of the software product. Areas to consider include those items and components associated with the released programs, the labeling of subcomponents such as document files and code modules (may be constrained by the support software), and the relationship between these two "levels" of identification. Special consideration must be given identifying computer programs embedded in [EP]ROM. If the software identification scheme is driven by the hardware identification scheme, describe the activities that support this relationship.

3.1.1 Baselines

Describe the relationships between the Functional, Allocated, and Product Baselines to the software development phases. Normally, the Functional Baseline is related to system requirements, and the Allocated Baseline is based on the software requirements. The Product Baseline is a version of the

released software. Identify any other baselines necessary to support to this project. Define the requirements to verify each baseline.

3.1.2 SCM Libraries

Describe the scope of SCM libraries: Personal Library, Project Library, and Release Library. Identify the format, location, documentation requirements and access control for each. On single developer projects there may be no difference between the Personal and Project Libraries. However, there should always be a distinct Release Library. Describe the relationship, if any, to the (Sandia) Drawing System.

3.2 Configuration Control

Define what software will be controlled (what is the lowest control element or smallest entity that will be controlled: what are the deliverable products or configuration items (CIs) that will be released as separate entities) and what mechanism(s) will be used to facilitate change control , e.g., Software Change Request (SCR), Version Description Document. For software utilizing the Engineering Release System, use of the Advance Change Order (ACO), Final Change Order (FCO), and Complete Engineering Release (CER) could be addressed.

Describe the procedures used to make changes to known baselines. Identify the level of authority necessary for controlling changes to each baseline and each SCM library. This may vary over the life cycle of the project. The change authority level necessary to make changes before a baseline is established is usually less than after it has been approved. Likewise, changes to components in a Personal Library are usually self-approved by the developer while a change to a component in the Project Library that two or more people are working out of requires approval by some higher level authority (e.g., an SCCB.)

Define what OTS software will be controlled. This should include operating systems, compilers, and purchased test or emulation software.

3.2.1 Software Change Request / Requesting Changes

Describe the methods and activities used to process change requests. Identify any differentiation based on whether the change originated during test or operational use, or whether the change is a correction, enhancement or a conversion (an adaptation to a new environment - hardware/operating system, etc.) . Also, identify the information required for approval of a change and how the change will be implemented throughout the documentation (requirements specification, design documentation, test plan and procedures.)

3.2.2 Functions of the Configuration Control Board

Stipulate the scope and authority of the review board(s) and the operational procedures to which the board(s) will subscribe . The size of the project will determine the size of the CCB and the representation that the CCB requires. For small projects, the CCB may be a very informal group and may not consist of more than a couple of people. In this case the roles may be little more than a coordination effort between a developer and a user or next assembly developer. On larger projects,

the CCB may be involved in coordinating all the technical work performed by groups of developers. There may be multiple CCBs if the project complexity warrants it. The CCB of a project may have to interface with other change authorities due to interoperability issues. External requirements (identified in 2.3, Applicable Policies, Directives, and Procedures) may require that explicit functions be performed by the CCB. Those would be detailed here.

3.2.3 Evaluating and Implementing Changes

Define the engineering analysis required to determine the impact of proposed changes and the procedures to review the analysis. This may change as development progresses. Describe the activities and processes for verifying and incorporating changes. Specify any minimum information necessary for a change package to be complete. Identify the activities for releasing new baselines.

3.3 Configuration Status Accounting

Identify what types of information need to be reported and who the intended audience is. Status of change requests should be available to developers and managers. Developers would need to know the content of any particular baseline, especially the revision level of each module in the operational program. Transaction records may be necessary periodically to recover from mistakes. Managers may also want status accounting data to track progress against a defined Software Development Plan.

3.4 Audits and Reviews

Describe the review activities that are normally performed as an oversight function. They are usually designed to ensure that the developers have done all their work in a way that will satisfy internal and external obligations. They could also include areas such as operation of the SCM libraries, adherence to SCM procedures specified in this plan, performing a Physical Configuration Audit (PCA) — determining that all the items identified as being part of configuration are present, or performing a Functional Configuration Audit (FCA) — an acknowledgment that each item was tested or inspected to verify that it satisfied the functions defined in the specifications or contract for which it was developed.

3.5 Interface Control

The discussion should concentrate on the software-software and software-hardware interface elements for most plans. Describe or reference any agreements, control mechanisms, or other procedures that address the interfaces important to this project. These are the activities that ensure changes to the CIs of this project are coordinated with changes to interfacing items outside the scope of this plan or external to this project.

If this is a large project with many diverse groups, organizational interface elements should be included as well as life cycle phase transition interfaces. If organizational interfaces are addressed, the activities supporting that structure would also be detailed here.

3.6 Subcontractor/Vendor/Supplier Control

Describe any requirements placed on vendors or suppliers external to this project that relate to the need for insight into their life cycle process configuration management approach. Describe the SCM activities associated with incorporating any software developed outside the scope of this project into this project. This may include defining how the supplier will be monitored for compliance to contractual agreements and any supplier participation in change activities.

> *This section or Section 2 should describe configuration control procedures for integrating changes to supplier software and testing/verification of components dependent on that software. For example, how is software re-verified when a new compiler version is released?*

Discuss how any proprietary items will be handled for security of information and traceability of ownership. Even if the software is acquired as OTS software, there should be a description of how the software will be received, tested and placed under SCM. As a minimum, any acquired support software should be placed under SCM for version identification.

3.7 Records Collection and Retention

> *Note: This section used to be Section 6 in the old Std 828-1983. The new Std 828-1990 does not explicitly address this area. It has been added to Section 3 in this template because the author feels it is a necessary element of any Software Configuration Management Plan.*

Define the information that will be kept, the location of the information, and the length of time that it will be kept. As a minimum this should consist of copies of released material that provide backup and disaster protection for the life of the project. Other information may include change history, test and approval records, CCB proceedings, audits, reviews, and status reports. Cost, liability, and warranty considerations will help in deciding how much information to keep.

4. Schedules — SCMP Implementation

> *Note: The contents of this section was previously in Section 2 of the old Std 828-1983. The new Std 828-1990 has moved Implementation (now Schedules) to this section.*
>
> *(It is better not to put dates in Section 4. Any schedules with actual dates should be included as an attachment. A chart overlaying SCM activities onto project management major milestone time line is one effective representation of the SCM schedule. Be sure to focus on major configuration control events.)*

4.1 Configuration Control Board

Specify in terms of the life cycle or milestones when the CCB will be established, when it will accomplish major tasks such as approving a complete baseline or release, and when significant changes in the structure of the CCB are expected to occur.

4.2 Baselines

Describe any dependencies concerning other project activities, other SCM activities, and the timing of establishing and releasing baselines.

4.3 Change Control

Specify when changes must begin to be controlled and when any increase in the formality of that control will be implemented. For example, changes may be allowed without formal review prior to establishing some intermediate baseline of detailed design information; changes may be required to be individually inspected during system test.

4.4 Status Reporting, Reviews, and Audits

Identify when status information on the developing software will be done. Identify the points in the life cycle when the audits and/or reviews identified in 3.4 are to be performed. As with 4.2 above, describe any dependencies with other activities. Reimbursable programs may have to provide specific information at regular or predefined intervals.

5. Resources

Note: The contents of this section was previously found in Section 4 and Section 2.4 of the old Std 828-1983. The new Std 828-1990 has combined them into this section.

5.1 Tools, Techniques, and Methodologies

Identify any software tools, manual procedures, and methodologies that are SCM-specific or are general aids to the SCM activities identified in this plan. These can include control of the SCM Library structure, access control, tracking of documentation and code, baseline system generation, change processing, problem tracking and control, status reporting, and archiving and retrieval tools.

5.2 Personnel

Identify the personnel necessary to perform SCM on this project, their function(s), and the activities in which they will be involved. This is more of a count of the personnel (Full Time Equivalents or FTEs) or naming of individuals with special skills). On small projects, this was probably addressed in Section 2.2. On large projects, Section 2.2 may have only discussed responsibilities at the organizational level. When the number of people required is not clearly defined in Section 2.2, it should be identified here.

5.3 Training

Identify any training requirements necessary to implement SCM on this project. This may include training in SCM techniques and tool-specific training.

6. Plan Maintenance

Identify the Owner of this plan and how often it will be updated (or reviewed for possible update). Specify how changes and suggestions to this plan are to be communicated, and how those changes are to be evaluated and approved.

Distribution:

0300 R. L. Schwoebel
0326 M. A. Blackledge
0326 D. E. Peercy (5)
0326 E. H. Tomlin (20)
0336 G. E. Dahms
0363 D. A. Clements
0364 J. P. Martin
1204 P. L. McAllister
1424 J. L. Tomkins
2273 M. A. Tebo
2314 R. G. Husa
2337 M. C. Kidd
2545 S. L. Trauth (5)
2615 D. D. Neidigk
2615 P. A. Trellu
2725 B. R. Ahrens (5)
2818 O. H. Bray (5)
3827 B. L. Straba (250)
5012 B. P. Gaude (5)
5363 R. A. Van Cleave
5711 J. P. Franklin
5931 G. F. Quinlan
6418 R. M. Summers (5)
6413 D. T. Chanin
6422 T. J. Heams
6522 G. C. Giesler
8117 F. J. Cupps
9227 L. M. Grady
9232 M. T. McCornack
9545 L. M. Desonier

8523-2 Livermore Library (2)
3141 S. A. Landenberger (5)
3913-2 L. G. Byrum
For DOE/OSTI (8)
3151 G. C. Claycomb